Vasco da Silva Brito

Licenciado em Ciências da Engenharia Eletrotécnica
e de Computadores

Fault Tolerant Control of a X8-VB Quadcopter

Dissertação para Obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: Luís Filipe Figueira Brito Palma, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente:	João Almeida das Rosas, Professor Auxiliar, FCT-UNL
Arguente:	Alberto Jorge Lebre Cardoso, Professor Auxiliar, FCT-UC
Vogal:	Luís Filipe Figueira Brito Palma, Professor Auxiliar, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2016

Fault Tolerant Control of a X8-VB Quadcopter

To my Parents

Acknowledgments

The realization of this thesis represents the end of an adventure, one more important step in my life. I would like to thank all who participated directly and indirectly to fulfill this journey with success.

First of all, I would like to thank my advisor Professor Luís Filipe Figueira Brito Palma for the opportunity, transmitted knowledge, patience and friendship. I would also like to thank him for the unconditional support and confidence, as well as for the provided working condition and tools.

I would also like to show gratitude my colleagues and friends, Afonso Maria, André Quintanova, Alexandre Brito, Guilherme Almeida, Liliana Sequeira, Rui Barrocas, Rui Antunes, Tiago Santos, Yaniel Barbosa among others for the unconditional support and friendship. Similarly, to my mentees, Miguel Pita, Luís Alves and Marcos Estrela, special thanks for the effort made during the programs.

Special thanks to my friends Ana Areias, Catarina Martins, Ricardo Bettencourt and Samuel Meneses for helping me unwind from time to time when I needed.

A profound acknowledgment to my family for the strength, protection and dedication provided along these years. To my parents, Carlos and Alda Brito for helping me grow into what I am today and for the sacrifices made in offspring of my education and professional formation. I would also like to recognize my uncle Paulo Brito, my aunt Benvinda Brito and my cousins Carolina and Ana Sofia Brito for having me every Sunday for lunch (sometimes for diner too), for our constructive discussions about every kind of subject and for the good advices.

Last but not least, many thanks to Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa for allowing me to enlarge my culture and knowledge of the world over the last 6 years and more recently to Center of Technology and Systems of Uninova for having me as a research collaborator over the last year.

Abstract

In this dissertation new modeling and fault tolerant control methodologies of a quadcopter with X8 configuration are proposed; studies done to actuators faults and possible reconfigurations are also presented.

The main research effort has been done to design and implement the kinematic and dynamic model of a quadcopter with X8 configuration in Simulink®. Moreover, simulation and control of the quadcopter in a virtual reality world using Simulink3D® and real world experimental results from a quadcopter assembled for this purpose.

The main contributions are the modeling of a X8 architecture and a fault tolerant control approach. In order to show the performance of the controllers in closed-loop, simulation results with the model of a X8 quadcopter and real world experiments are presented.

The simulations and experiments revealed good performance of the control systems due to the aircraft model quality. The conclusion of the theoretical studies done in the field of actuators' fault tolerance were validated with simulation and real experiments.

Keywords: quadcopter, modeling, fault tolerance, fault tolerant control, kinematic and dynamic systems, Arduino, Matlab®, Simulink®.

Resumo

Nesta dissertação propõem-se novas metodologias de modelização e controlo tolerante a falhas de um *quadcopter* com configuração X8; apresentando-se estudos realizados com falhas em atuadores e estratégias de reconfiguração.

O maior esforço de investigação foi despendido na conceção e implementação dos modelos cinemático e dinâmico de um *quadcopter* com a configuração X8 em Simulink®. As tarefas posteriores foram a simulação e o controlo do *quadcopter* num mundo de realidade virtual usando Simulink3D® e a obtenção de resultados experimentais do mundo real a partir de um *quadcopter* montado para esse fim.

As principais contribuições são a modelação de um *quadcopter* com arquitetura X8 e uma abordagem de controlo tolerante a falhas. A fim de mostrar o desempenho dos controladores em anel fechado, apresentam-se resultados das simulações com o modelo de um *quadcopter* X8 e resultados experimentais no mundo real.

As simulações e as experiências revelaram um bom desempenho dos sistemas de controlo devido à boa qualidade do modelo proposto para a aeronave. As conclusões do estudo teórico realizado na área de tolerância a falhas nos atuadores foram validadas em simulações e em testes reais.

Palavras-chave: *quadcopter*, modelação, tolerância a falhas, controlo tolerante a falhas, cinemática e dinâmica de sistemas, Arduino, Matlab®, Simulink®.

Notations and Symbols

Notations

ARX: auto-regressive linear model with exogenous input

CCW: counter-clock wise

CW: clock wise

DC: direct current

ESC: electronic speed controller

FDD: fault detection and diagnosis

FTC: fault tolerant control

LiPo: lithium polymer

MRAS: model reference adaptive systems

PID: proportional-integral-derivative

PSO: particle swarm optimization

PWM: pulse wide modulation

RPM: rotations per minute

SMC: sliding mode controller

SPE: solid polymer electrolyte

STR: self-tuning regulator

TDM: time division multiplexing

BLDC: Brushless DC motors

Symbols and Operators

Variables and scalars are represented by small letters in italic, (ex. $x(t), a, ...$). Matrices are represented by capital letters in bold, (ex. $\boldsymbol{M}, \boldsymbol{M}(:,:,k), ...$). Vectors are represented in small letters, in bold, (ex., **x**(:,k), **y**, …).

t	Continuous time variable
k	Discrete time variable
$x(t)$	Continuous time signal at instant t
$x(k)$	Discrete time signal at discrete time k
T_s	Sampling time
g	Gravity acceleration
m	Body mass
e	Control error
v	Temporary control actuation
u	Control saturated actuation
U_n	Sum of motors' forces
ϕ, θ, ψ	Roll, pitch and yaw, respectively
f_n	Motor's force
ω_n	Motor's rotation speed (rad/s)
m	Mass (Kg)
l	Length (m)
h	Height (m)
r	Radius (m)
sat()	Saturation function
mse()	Mean square error function
var()	Signal variance function

Table of Contents

List of Figures

List of Tables

List of Tables - Introduction

Chapter 1

1 Introduction

1.1 Motivation

The operation of aircrafts requires a good amount of expertise and, in today´s quest of finding the cheapest solution, the need of hiring an operator can increase the costs significantly. The current growth of interest in unmanned aircrafts has been changing the companies' world. The major problems regarding these kinds of subjects are the reliability and security of the aircrafts and the people. Over-actuated systems, as well as fault detection and diagnosis, are fascinating and complex topics, representing major research areas. Their application to unmanned aircrafts is just the tip of the iceberg involving different areas such as control, soft computing, digital image processing and much more.

This research is related with the fault diagnosis and fault tolerant control implemented in an over-actuated system. The control challenge and the security of aircraft and people led the author of this dissertation to work in this research field.

1.2 Main Goals and Contributions

Until now, the main research effort in multirotors control has been mostly focused in the attitude and trajectory control, considering that the aircraft' structures do not fail.

The main goal of this dissertation is to focus on the detection of failures on the aircraft's actuators corresponding to a quadcopter's fault, designing fault tolerant controllers and apply them to a real world situation. Another objective is to assemble the quadcopter X8 design, in order to posteriorly apply the algorithms of fault tolerant control. Moreover, to create a model that represents the dynamic and kinematic of the aircraft and, with this, design the control algorithms. An additional intent is to develop a virtual reality world where the model dynamics and kinematics could be easily observed and explained to people.

This dissertation will follow the subsequent workflow diagram (Figure 1.1) where parallel activities are present.

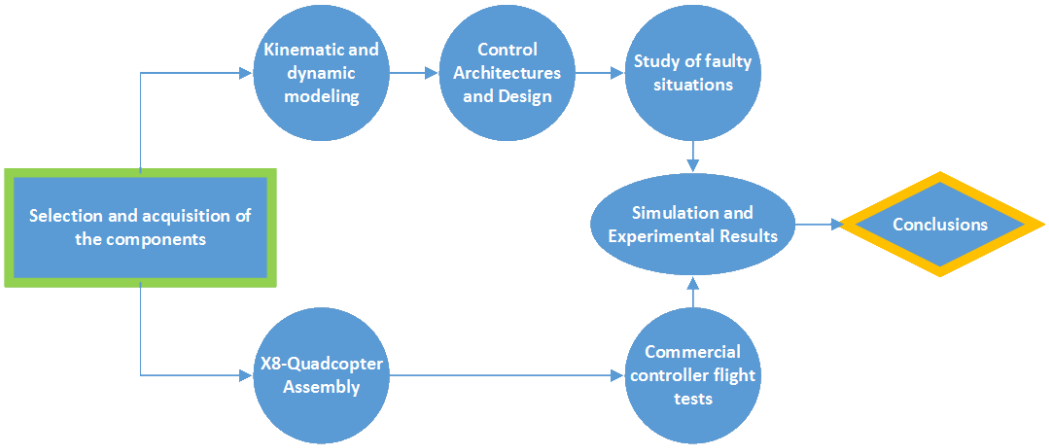


Figure 1.1 - Thesis workflow diagram.

The main contribution is the use of fault detection applied to a real world aircraft using algorithms of control allocations, in order to allow an over-actuated multirotor to fulfill a mission under faulty conditions. Another contribution is the development of a kinematic and dynamic model allowing to better understand the dynamic behavior of aircrafts of this type and able to simulate faults/failures. In addition to these contributions, the model will contribute to the educational area, allowing students learn about aerodynamics.

1.3 Thesis structure

This dissertation is organized in 5 chapters, including this one. The organization is as follows:

In Chapter 1, the motivation, the main goals, and contributions of the dissertation structure are presented.

Chapter 2 contains the state-of-the-art with background concepts used in the research, which come from diversified areas. These concepts are presented and discussed in this chapter.

Chapter 3 shows the research development, from the mathematical modelation of the dynamic and kinematic of the aircraft, through the motors and propellers modelation, until the final and most important topic arrives - the fault tolerant control technique developed.

In Chapter 4, simulations and experimental results of the fault tolerant control algorithms and other flight trials are presented here as well as a comparison between methods.

At the end, the conclusions and the future work appears in Chapter 5.

Chapter 2

2 State of the Art

2.1 Introduction

The state of the art will present and discuss all the main concepts used in this work. The ground work for this research came from different fields of study. Concepts can derive from DC motor or propellers to the physical modulation of a full functional quadcopter.

Notions like the origin of multirotors, DC motors, propellers as well as Proportional Integral Derivative (PID) controllers, Fuzzy logic, Sliding Mode Controllers (SMC), Fault Detection and Diagnosis (FDD) and over-actuated systems are briefly reviewed here.

It is beyond the objective of this document to give a complete treatment of all of these fields. However, to clarify any terms which may be misunderstood, some definitions will be presented here, since the scientific and technical community sometimes may not be consensual in some terminologies.

2.2 Multirotors

2.2.1 History

Louis and Jacques Breguet invented the first quadrotor in France, with the supervision of Charles Richet, in 1907. It was the first piloted aircraft able the lift vertically. Originally, it was called gyroplane, now adapted to helicopter or multirotor. The design was very simple, as we can see in Figure 2.1. This was the first known attempt to build a quadrotor and it is known to have actually flown several times (Young, 1982).

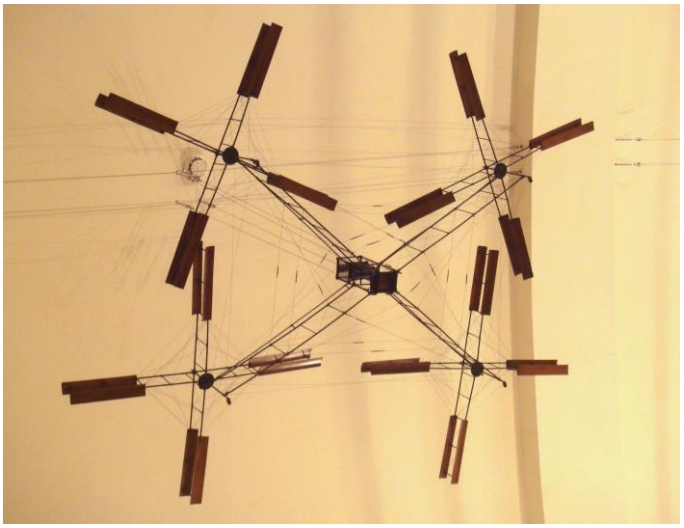


Figure 2.1 - Breguet-Richet Gyroplane.

Following the previous successful experiment, in 1920, Etienne Oehmichen made his attempt with the multirotor design. After about six try outs, he finally arrived at the "Oehmichen No.2" (Figure 2.2). The helicopter had small rotors spinning in the opposite direction from the large lifting rotors, which led to the invention of the tail rotor that we today know in common helicopters (Oehmichen, 1924).

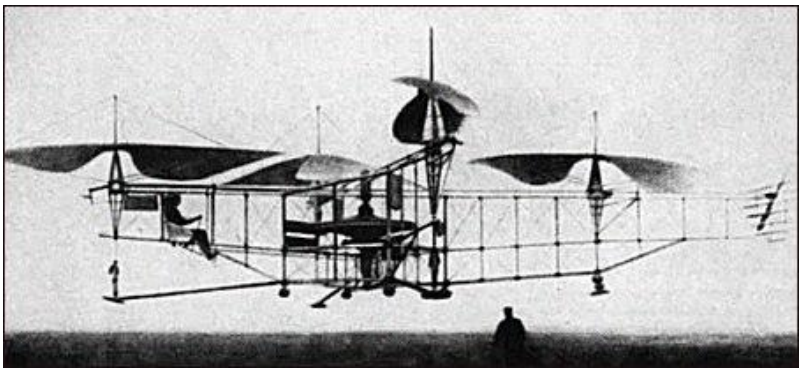


Figure 2.2 - Oehmichen No.2 aircraft.

The next important step in quadcopter history was a prototype made by Convertawings, Inc. in 1956 called Model A Quadrotor (Figure 2.3). The design had two engines driving four rotors, no tail rotor was needed, and the control was made by varying the thrust between the rotors, like nowadays quadcopters. It flew successfully many times and was the first quadrotor design able to forward flight reliably. Unfortunately, due to lack of commercial and military needs, the project was abandoned (Convertawings Inc, 1956).

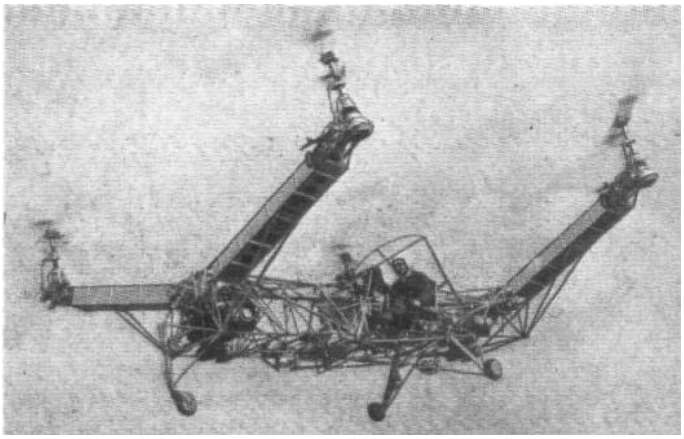


Figure 2.3 – Convertawings, Inc. Model A Quadrotor.

After these first attempts, the world of multirotors continued evolving. Many other scientists and investigators have been developing knew aircrafts with different ends, such as military and commercial use.

2.2.2 DC Motors

The purpose of the electric motors is to convert electrical energy into mechanical energy. This is possible by interacting magnetic fields - one stationary and the other moving. DC motors are the simpler and older motors around, invented in the early 1820s. The design was possible after Oersted, Gauss and Faraday discovered the electromagnetic principles in the early 1800's. In October of 1821, Faraday successfully confirmed the conversion of electrical energy into movement (Figure 2.4); the demonstration was published in the "Quarterly Journal or Science Literature, and the Arts" in 1822 (Royal Institution of Great Britain, 1822), being recognized as the inventor of the electrical motors concept (Qadir, 2013).

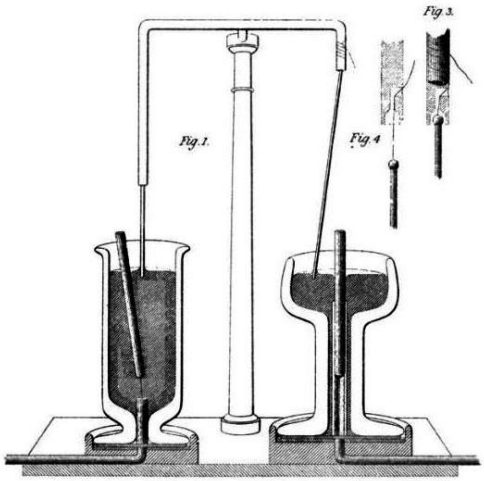


Figure 2.4 – Faraday's electromagnetic experiment, 1821(Faraday, 1822).

Several years after Faraday's experiment, Joseph Henry improved the design by building an electromagnet rotor that reversed automatically the polarity by its motion as pairs of wires made contact alternately with two cells; two permanent magnets create a flux that alternately attract and repel the electromagnets, making them go back and forth at 75 cycles per minute. Although it was only a lab experiment, it was more mechanically useful than Faraday's early research. Joseph Henry's design also showed that it was feasible to develop electrical generators and motors (Qadir, 2013).

Not long after Henry's tryouts (1832), William Sturgeon invented the commutator, alongside the first continuous rotation electric motor. The design was still simple, containing most of the elements of modern DC motors (Qadir, 2013).

After long years of evolution regarding DC motors, Brushless DC motors (BLDC) appeared in 1962, developed by T.G. Wilson and P.H. Trickey, which was called "CD Machine with Solid State Commutation" (Wilson & Trickey, 1962). In the beginning, the technology to make such a motor practical for industrial use did not exist. Only after the arrival of powerful magnet materials and high voltage transistors, in the early 1980's, was the use of BLDC a possibility (Qadir, 2013). The key component differing BLDC and their predecessor's most common used DC motors was the loss of an element, as the name suggests, the brushes. Without these, the efficiency of the motor increases, as well as the longevity, since the friction between the rotor and the brushes no longer exist. It also has a major advantage - because there is no need of sliding contacts, brushes or excitation windings, the BLDC react with promptness to variations in current, making them better at dynamic performance (Yedamale, 2003).

The exclusion of the brushes makes the motors easy to assemble and repair, as well as modifying the motors' construction, making them smaller and lighter. This progress also improves the speed contrasted with torque characteristic and can reach high speeds with less noise. Because of the construction design, the BLDC have high electrical efficiency (Yedamale, 2003) being the perfect match for drone use.

2.2.3 Propellers

The principal of a propeller is to convert rotation motion into propulsive force. The first known sketch of a propeller applied to airlift is from the Italian scientist Leonardo da Vinci, circa 1490 (Figure 2.5 b). It was a human-powered helicopter with no real-world application (Heatly, 1986). The propeller design in Leonardo's drawing was inspired by Archimedes' water screw invention (Figure 2.5 a).

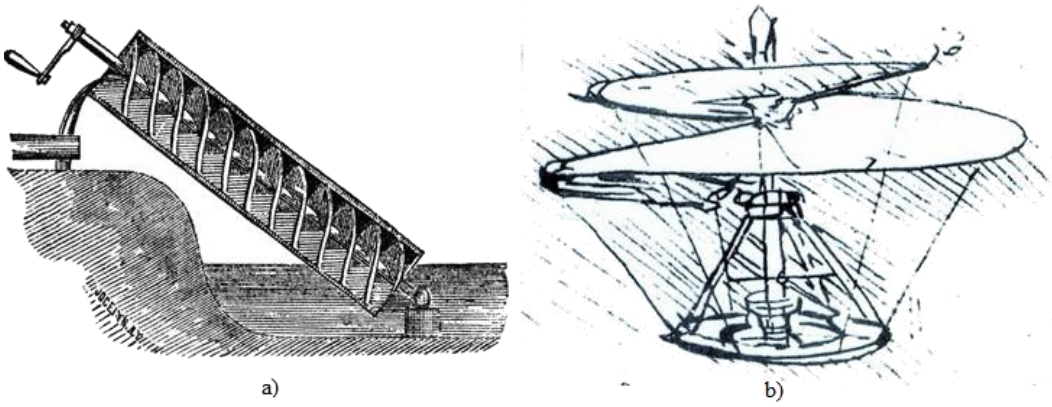


Figure 2.5 - a) Archimedes' screw; b) Leonardo da Vinci's draw (Keele, 2014).

The Wright Brothers engineered the aircraft propeller that we know today. By conducting experiments with wings in wind tunnels, they realized that propellers had essentially the same shape and behavior as wings (Figure 2.6). They also realized that a twist along the length of the blade was needed in order to keep the angle of attack uniform alongside all the extension of the propeller (Crouch, 2004).

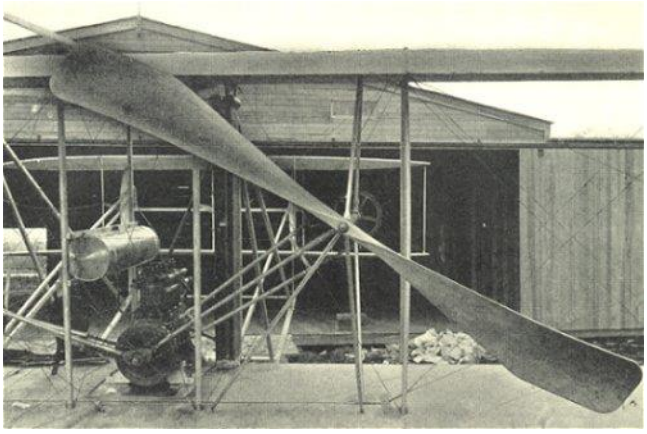


Figure 2.6 - Wright Brothers' propeller (Kaempffert, 1911).

The design of propellers has the angle of attack as a major effectiveness factor. As the Wright Brothers discovered, the angle of attack needs to be constant along the length of the propeller's blade, eliminating the possibility of stall. Another significant factor of effectiveness is the pitch. It is defined by the distance travelled in one revolution. These two factors allow the measurement of the efficiency of propellers (Federal Aviation Administration, 2008).

2.2.4 Energy Power Source

Lithium Polymer batteries follow the same path of lithium-ion - a lithium-metal cells that underwent most of its research during 1980s. The first commercial cylindrical Li-ion cell, in 1991, by Sony, was a significant milestone in the field. Other packaging techniques evolved after that point, including the format of the most common used batteries applied to drones known as LiPo (Lithium Polymer). The primary difference between lithium-ion and LiPo batteries is that the first one uses a lithium-salt electrolyte held in an organic solvent and the second one uses a solid polymer electrolyte (SPE) (Manuel Stephan & Nahm, 2006).

There are three types of SPEs: dry SPE, gelled SPE and porous SPE. Michel Armand first used the dry SPE in prototype batteries around 1978 (M.B. Armand, J.M. Chabagno, 1978). Later (1985), other companies used dry SPEs like ANVAR and Elf V Acquitaine from France, as well as Canadians' Hydro Quebec. The gelled SPEs were used some years later, around the 1990s, by several organizations like Mead and Valence from The United States and GS Yuasa in Japan. Although the American Bellcore started commercialization of rechargeable LiPo cells using porous SPE, the business was not successful (Murata, Izuchi, & Yoshihisa, 2000). The historical evolution can be observed in Figure 2.7.

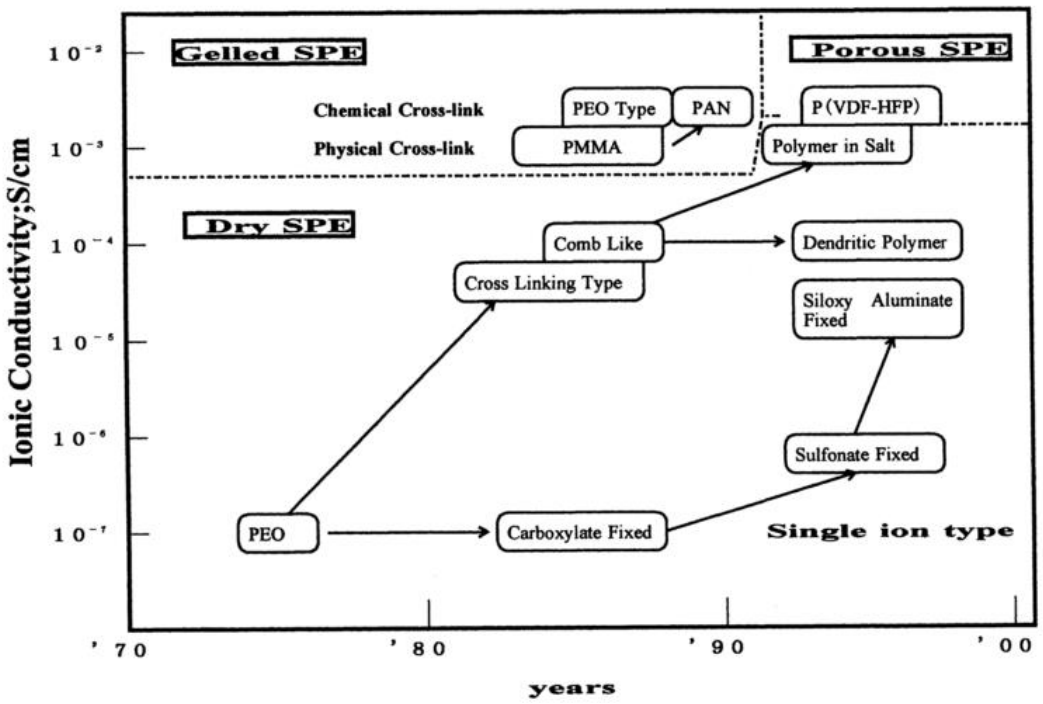


Figure 2.7 - History of ionic conductivity improvements (Murata *et al*., 2000).

The work principle of LiPo cells is rather simple and similar to lithium-ion cells. There is an intercalation and de-intercalation of lithium ions from the positive and negative electrode, with liquid electrolyte offering a conductive mean. To avoid the electrodes from making contact directly, a microporous separator is used between them allowing only ions and not electrode particles to travel from one side to the other (Scrosati, 2001).

2.3 Control Approaches

2.3.1 PID

A Proportional Integral Derivative controller is the most commonly used way to solve practical control problems. While proportional and integral controllers were used before, the PID as we know today was developed in the 1930s with pneumatics (Karl Johan Åström & Hägglund, 2006).

The generic PID controller algorithm can be written as:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \quad (2.3.1)$$

where $u(t)$ is the control signal, $e(t)$ is the error ($e(t) = r_{ref}(t) - y_{sys}(t)$) and the three control parameters are: the proportional gain K, the integral time T_i and the derivative time T_d (Karl Johan Åström & Hägglund, 1995).

2.3.1.1 Proportional Action

Regarding the control action, when it is simply a proportional control, the equation (2.3.6) will reduce to:

$$u_p(t) = Ke(t) + b \quad (2.3.2)$$

The control is simply proportional to the control error. The variable b is a bias or a reset. When the error reaches zero, the control action becomes $u_p(t) = b$. Bias b often takes the value $(u_{max} + u_{min})/2$, but in some cases can take another value, in order for the stationary control error to be zero at a desired set point (Karl Johan Åström & Hägglund, 1995). An example of the proportional controller in a closed-loop system is represented in Figure 2.8 where the proportional control is not enough for the process to reach the setpoint in steady state. It also makes the closed-loop system oscillatory as K increases (Karl Johan Åström & Hägglund, 2006).

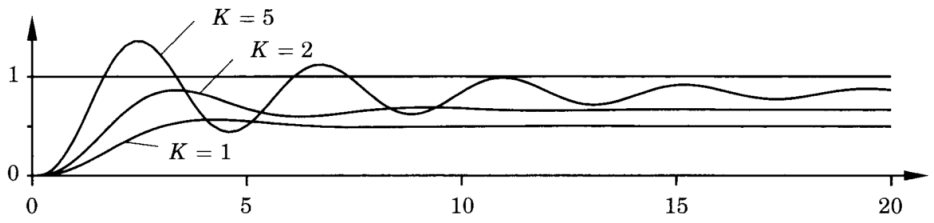


Figure 2.8 - Closed-loop system simulation with proportional control. The transfer function is $G(s) = (s+1)^{-3}$. The set point applied is y_{ref} =1. Adapted from Åström & Hägglund 2006.

2.3.1.2 Integral Action

As we have seen in the proportional action simulation, the proportional gain was not enough to reach null control error in steady state. The main goal of the integral action is to guarantee that it does happen, assuring the lowest value between the process output and setpoint in steady state. The integrator is the sum of the instantaneous error over time. However, there will always be a sum of error, no matter how small it is (Karl Johan Åström & Hägglund, 2006).

The process of calculating the integral action can be given as follows:

$$u_i(t) = K\left(\frac{1}{T_i}\int_0^t e(\tau)d\tau\right) \tag{2.3.3}$$

The term accelerates the process response towards a disturbance. The disadvantage of the integral component is that it responds to accumulated errors from the past, which can cause the present value to overshoot. Anti-windup algorithms are the way to counteract this effect such as Set-Point Limitation, Incremental Algorithms, Back-Calculation and Tracking, among others (Karl Johan Åström & Hägglund, 2006).

When the integral and the proportional terms join, we end up with a Proportional Integral (PI) controller that can be written as:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\right) \tag{2.3.4}$$

The same closed-loop system was tested using the PI controller. The proportional gain was set to K=1, in order to compare with Figure 2.8. The results of the PI controller are shown in Figure 2.9.

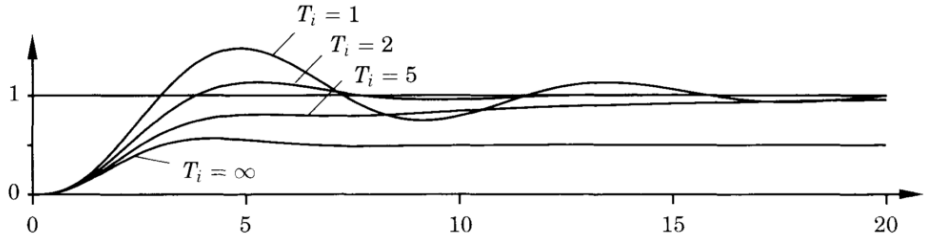


Figure 2.9 - Closed-loop system simulation with PI control. The transfer function is $G(s) = (s+1)^{-3}$. The set point applied is y_{ref} =1. Adapted from Åström & Hägglund 2006.

Figure 2.9 shows that in the case of $T_i = \infty$ we have only the proportional component - the same as in Figure 2.8. The steady state error is reduced when applying the T_i term. For large integration times, the closed-loop system response slowly moves to the setpoint and for small integration times, the response is faster but it is oscillatory (Karl Johan Åström & Hägglund, 2006).

2.3.1.3 Derivative Action

From the previous experiments, the setpoint in steady state was already reached but with an excess of overshoot and scarce damping.

It is now that the derivative term comes in. It predicts the closed-loop system's behavior, improving the stability and the settling time. The prediction is made by inferring the error by the tangent to the error curve (Karl Johan Åström & Hägglund, 2006). Because of the causality, some implementations of PID controllers have an additional low pass filter, in order to limit the high frequencies gain and noise (Ang, Chong, & Li, 2005).

With the integration of the derivative term we arrive at the equation (2.3.6). Testing the closed-loop system once again, with K=3 and $T_i = 2$ (Karl Johan Åström & Hägglund, 2006) the response was as in Figure 2.10.

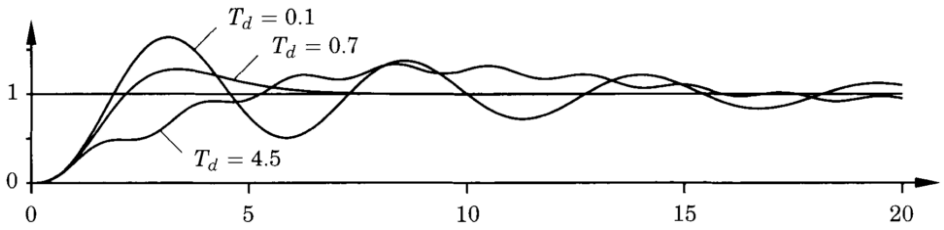


Figure 2.10 - Closed-loop system simulation with PID control. The transfer function is $G(s) = (s+1)^{-3}$. The set point applied is y_{ref}=1. Adapted from Åström & Hägglund 2006.

From Figure 2.10 one must conclude that by increasing the derivative time, the closed-loop system's response slowly increases the dumping until there is none. From a certain T_d, the dumping begins to decrease until the closed-loop system becomes oscillator.

2.3.2 Fuzzy

Fuzzy logic dates from 1965, first proposed by Lotfi Aliasker Zadeh in California. As L. A. Zadeh realized that the world is not set by deterministic rules but on an antagonistic way, it is set by some level of uncertainty. Concepts contain elements of subjectivity, like a car going fast or slow depending on the perception of the person driving (Santos, 2016). The idea of fuzzy is to translate common language into computer language, in order to compute the data (Zadeh, 1965). A common fuzzy control architecture is presented in Figure 2.11.

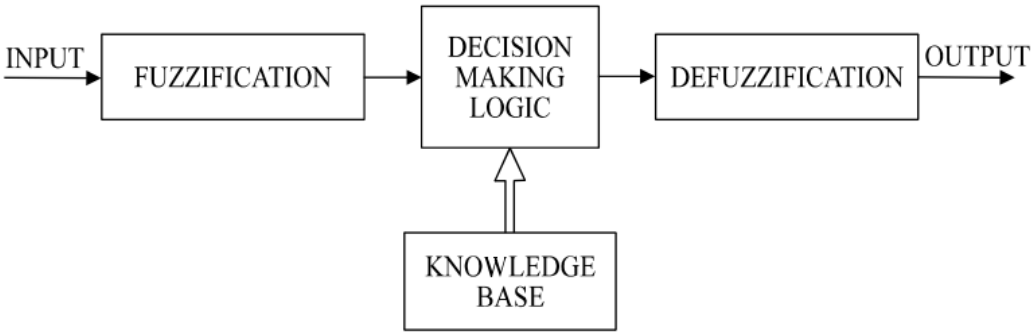


Figure 2.11 - Fuzzy controller architecture (Simoes, 2003).

2.3.2.1 Membership Function

The main difference between a crisp solution and fuzzy is the membership function. The membership function on a crisp solution is, for instance, [0, 1] or [High, Low], not offering a good model of the real world. On the other hand, the fuzzy logic moderates this condition making the existence of elements simultaneously holding nonzero membership grades possible. Because of this, fuzzy offers a more flexible solution than the crisp approach (Zadeh, 1965).

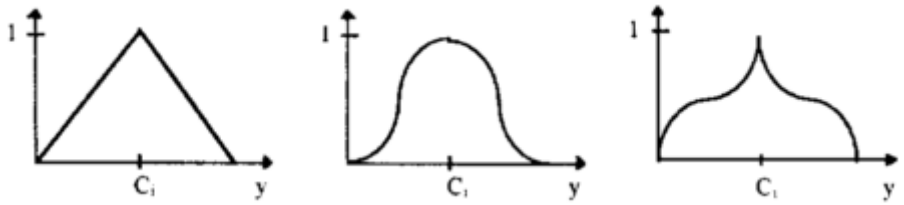


Figure 2.12 - Example of membership functions (Rondeau, Ruelas, Levrat, & Lamotte, 1997).

2.3.2.2 Fuzzification

Fuzzification is where the data is partitioned into fuzzy subspaces, which can overlap on each other (creation of membership functions Figure 2.12). In order to obtain the result of the fuzzification, there must be a membership function associated individually to each subspace. The result of this operation is a membership grade that will be processed in the next step (Rondeau *et al.*, 1997).

2.3.2.3 Inference

Here, the results of the fuzzification are aggregated and the strength contributed by each triggered rules are computed. The aggregation is usually obtained by two methods: union (max), or the disjunction (min). There are several other techniques to perform this aggregation process, which may have a decisive role on the final output (Ross, 2009).

2.3.2.4 Defuzzification

Defuzzification is a process where the values obtained after the inference are translated into whatever the computer or person understands. There are more than a few methods of defuzzification, such as centroid, mean-of-maximum and others (Rondeau *et al*., 1997).

2.3.3 SMC

Sliding mode control emerged in the late 1960s, designed by Vadim Utkin in the Soviet Union. This control architecture is common amongst variable structure systems where dynamic changes to the systems or non-linear systems are present. The technique is very efficient when treating uncertainties related to model linearization, as well as external disturbances (Spurgeon, Sabanovic, & Fridman, 2004).

2.3.3.1 Sliding Mode Concept

The sliding mode perception is common in dynamic systems characterized by differential equations with state functions containing discontinuities. Usually, the example used of sliding mode is a second order relay system that can be found in most textbooks on nonlinear control (Utkin, 1992).

$$\ddot{x} + \mathrm{a}_2\dot{x} + \mathrm{a}_1 x = u \tag{2.3.5}$$

$$u = -\mathrm{M}\, sign(g), \quad g = \mathrm{c}\, x + \dot{x}, \qquad \mathrm{a}_1, \mathrm{a}_2, \mathrm{M}, \mathrm{c} \text{ are const} \tag{2.3.6}$$

The input takes two values, M and –M, and the discontinuities appears on the straight-line $g = 0$ in the state plane represented in Figure 2.13.

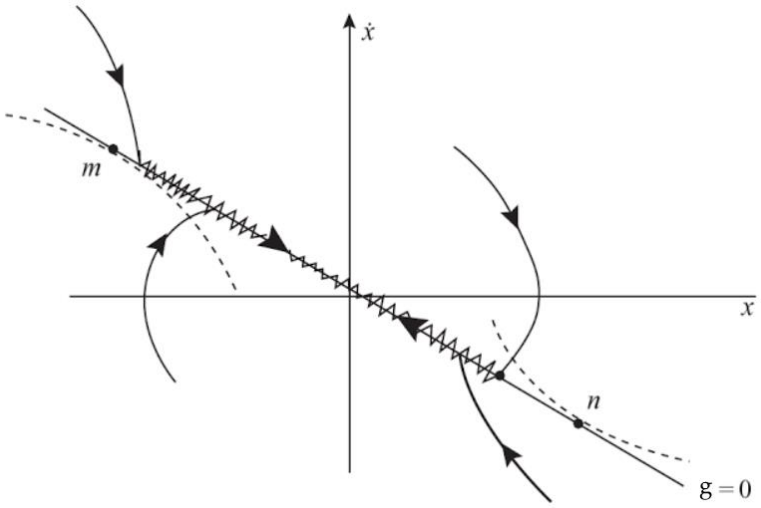


Figure 2.13 - Sliding mode in a second order relay system (Utkin, 1992).

Analyzing the state plane, the neighborhood segment on the switching line $s = 0$, the trajectories run in the opposite direction, leading to the appearance of a sliding mode along the line. Equation $\dot{x} + c\,x = 0$ may be deduced as the sliding mode equation. An important fact is that the equation has less order than the original system and the sliding mode does not depend on the plant dynamics, being described only by c. Variable structure systems have the sliding mode as principle operation mode (Utkin, 1992).

Sliding mode control approaches has its applications in the fields of control infinite-dimensional systems, control of systems with delay, sliding mode observers, parameter and disturbance estimators, adaptive control and Lyapunov function based design technics (Utkin, 2005), (Ferreira, 2016).

2.3.4 Digital Control

Digital control systems habits in a computer. The typical digital control systems interact with the plant using peripherical analog components. The processing of the controller equations, like the ones presented previously, that make the difference between analog and digital control. Some of the advantages of the digital control systems are: the reduced cost, low weight, and the power consumption (Levine, 2000).

In order to translate the signal information to the computer, discrete-time signals take place. These are defined by a discrete instant of time, normally regularly spaced time steps (Santina & Stubberud, 2005). The discrete-time signal is represented by a sequence of numbers usually named samples. In a discrete-time system, $f(k)$, k means the step index. An example of a discrete-time signal can be observed in Figure 2.14 where the representation of a simple line (equation(2.3.7)) is present and its discretization (equation (2.3.8)).

$$y(t) = x(t) \tag{2.3.7}$$

$$y(k) = x(k), k \in \mathbb{N} \tag{2.3.8}$$

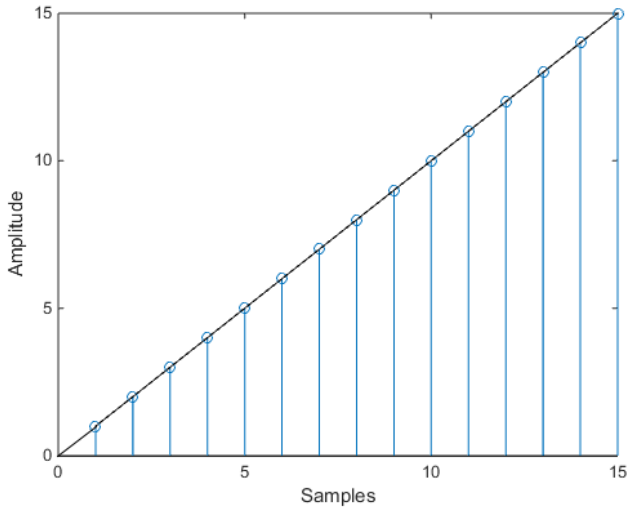


Figure 2.14 - Discretization example.

The main problems regarding the discretization of a continuous-time signal is the quantization error. Quantization is the process of rounding or truncating the input values. This should be considered when defining sample times, signal noise ration and so on.

2.3.5 Robust Control

Robust control designs are used to operate under uncertain parameters or disturbances in the closed-loop system. It aims to achieve robust performance in the presence of certain modeling errors. In contrast with adaptive control, robust control uses a static controller rather than adapting it. The controller is design to work assuming certain variables are unknown but bounded (Zhou, 1999) (Webb, Budman, & Morari, 1995).

The theory of robust control started around late 1970s and early 1980s, and since then the development of numerous techniques for dealing with bounded system uncertainties took place (Safonov & Fan, 1997).

McFarlane and Glover in Cambridge University developed one of the most important robust control techniques. The method is H-infinity loop-shaping. Such method minimizes the sensibility of the system over its frequency spectrum, guaranteeing that it does not diverge from the expected response once the system is disturbed (McFarlane & Glover, 1992).

In conclusion, robust control methods rely on a fixed control that tolerates changes of the system's dynamics, in order to assure the satisfaction of its goal under faulty situations. It is commonly known as a passive fault tolerant controller since its parameters are not changed according to the situation. Because of this, robust control methods are only used for restricted classes of changes in the system behavior. Additionally, robust control works under sub-optimal conditions for the nominal process since its parameters are fixed in order to get a trade-off between performance and robustness (Blanke, Kinnaert, Lunze, & Staroswiecki, 2006).

2.3.6 Adaptive Control

Adaptive control, as the name suggests, is a method of control that allows the controller to adapt its parameters. It can be seen as a switching controller where the switching signals are the controller's variables (J. Hespanha, 2001). The variation of the parameters tries to respond to dynamic changes in the process, as well as counteract the effect of disturbances. There have been several years of discussion regarding the difference between adaptive control and feedback control, since both counteract the effects of disturbances. In 1973, an IEEE[1] committee proposed a new terminology anchored in self-organizing control (SOC), performance-adaptive SOC and learning control system. Nevertheless, this idea was not implemented. (K. J. Åström & Wittænmark, 1994).

Nowadays, adaptive control is seen as a methodology for controlling systems with large modeling uncertainties, which implicate that Robust Control design tools may not be applicable (J. P. Hespanha, Liberzon, & Stephen Morse, 2003).

Adaptive control systems can be summarized as having two loops. As shown in Figure 2.15, one loop is a normal feedback with the plant and the controller, while the other loop is the parameter adjustment loop. There are several structures types of adaptive systems: gain scheduling, model-reference adaptive control and self-tuning regulators. These three types will be briefly described *infra*.

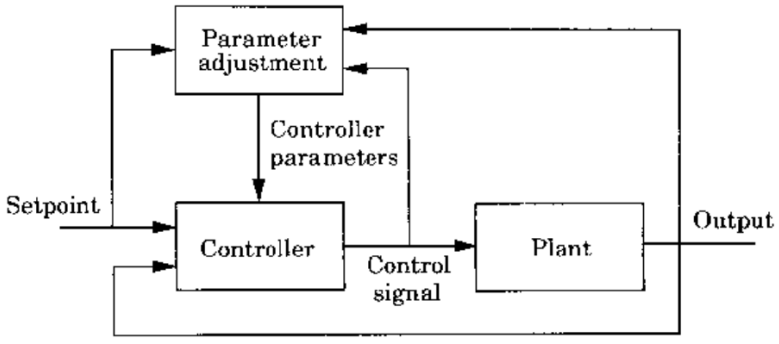


Figure 2.15 - Generic adaptive control diagram (K. J. Åström & Wittænmark, 1994).

[1] Institute of Electrical and Electronics Engineers, 1963.

2.3.6.1 Gain Scheduling

This method is applied when it is possible to find measurable system dynamic's variables that represent in a trustworthy way the changes in behavior. It is called gain scheduling because originally it was used to measure the gain and then change the controller parameter, that is, schedule them (K. J. Åström & Wittænmark, 1994). Figure 2.16 represents a structure diagram of a gain scheduling controller.

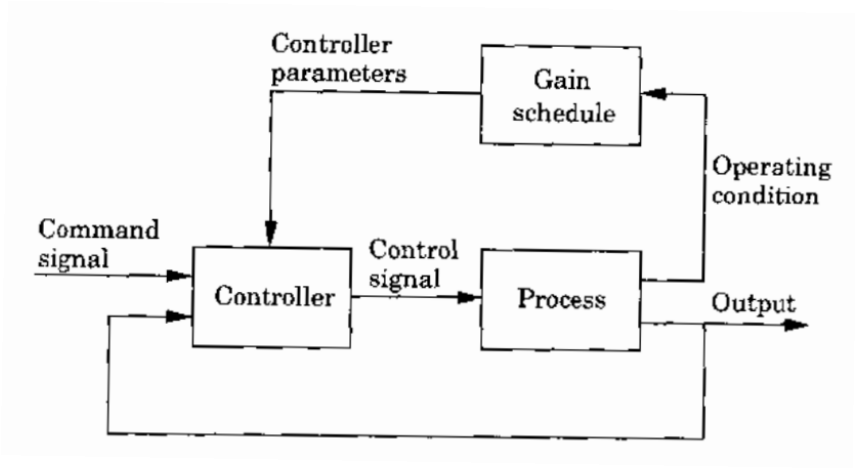


Figure 2.16 - Block diagram of a system with gain scheduling (K. J. Åström & Wittænmark, 1994).

These kinds of controllers are intimately close with the development of flight control systems. In the application mentioned in K. J. Åström & Wittænmark (1994), the two variables were Mach number and altitude, measured by air data sensors.

The gain scheduling is a very suitable technique to reduce the effects of parameter variations (K. J. Åström & Wittænmark, 1994).

2.3.6.2 Model-Reference Adaptive Systems

The model-reference adaptive system (MRAS) was first suggested in order to attend the performance specification in terms of a reference model. This method shows how the process output should ideally behave to an excitation signal (K. J. Åström & Wittænmark, 1994). The block diagram is shown *infra* in Figure 2.17.

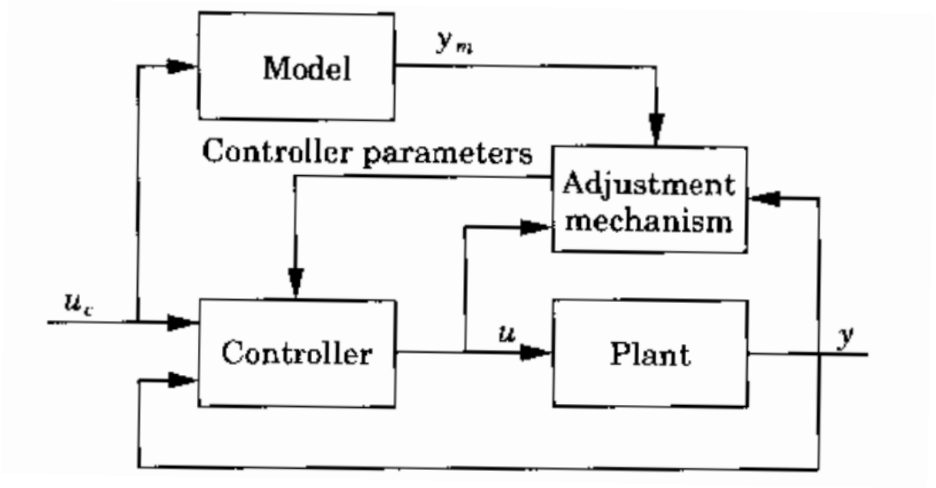


Figure 2.17 - Block diagram of a MRAS (K. J. Åström & Wittænmark, 1994)

The major problem with MRAS is finding the adjustment method in order to obtain a stable system, bringing the error to zero. This is not an insignificant problem. The first MRAS adjustment method was called MIT[2] rule, represented in the next equation (K. J. Åström & Wittænmark, 1994):

$$\frac{d\theta}{dt} = -\gamma e \frac{de}{d\theta} \tag{2.3.9}$$

In equation (2.3.9), $e = y - y_m$ represents the model error and θ is a controller parameter. The quantity $\frac{de}{d\theta}$ is the sensitivity derivative of the error and the parameter γ is the adaptation rate. To find the sensitivity derivate approximations are necessary.

The MIT rule can be considered a gradient method to minimize e^2 (K. J. Åström & Wittænmark, 1994).

[2] Massachusetts Institute of Technology

2.3.6.3 Self-tuning Regulators

The adaptive methods s*upra* are known as direct methods because the adjustment rules directly imply the changes in the controller parameters. A different method is encountered if an estimator of how the system parameters are updated is used to adjust the controller parameters (K. J. Åström & Wittænmark, 1994). The block diagram corresponding to this architecture is shown in Figure 2.18 - Block diagram of a STR (K. J. Åström & Wittænmark, 1994).

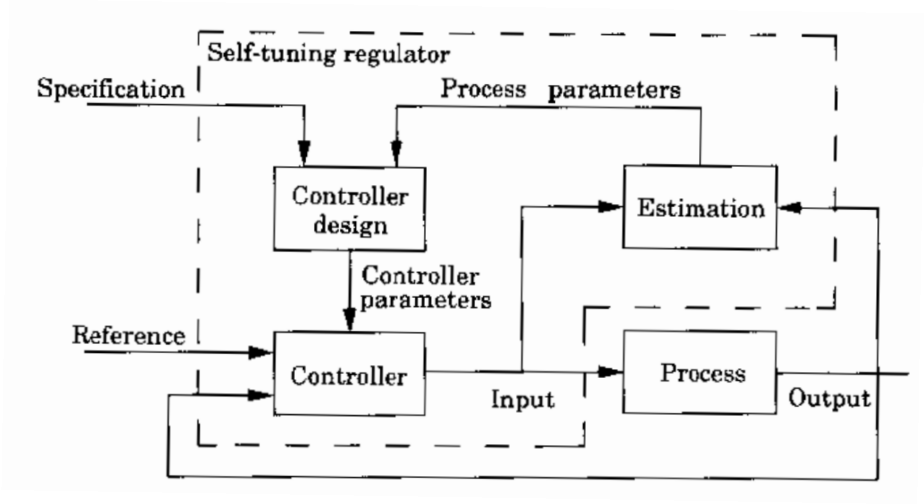


Figure 2.18 - Block diagram of a STR (K. J. Åström & Wittænmark, 1994).

The controller's name is meant to underline that the parameters are automatically tuned in order to obtain the closed-loop system proprieties desired. STR are a very flexible scheme, since the controller design and the estimation methods can vary in many ways (K. J. Åström & Wittænmark, 1994).

In STR control method, the process or controller parameters are updated based on real time estimations. These estimations are then used as the real parameters. This is called the certainty equivalence principle. The quality of the estimations can be measured in the same schemes and then used in the controller design. In some cases, the uncertainty is too large and one may choose a conservative method instead (K. J. Åström & Wittænmark, 1994).

Although the methods presented *supra* do not mention a supervisor (2.3.7 Supervised Control), all adaptive controllers need supervisory functions in order to operate well in a real world application (Hägglund & Aström, 2000). The following topic looks into this aspect.

2.3.7 Supervised Control

Supervised control is part of the everyday use of an industry. For instance, a human operator adjusting the parameters of a PID controller to account for changes in the environment, is a supervised control. Here, the human operator is a component of the feedback loop adjusting the dynamics using logic-based decisions (J. Hespanha, 2001).

The supervisor functions ensure that the controllers' action is only applied when the proper excitation is available. Situations like first phase of load disturbance response, bumpless transfer at parameter or controller changes and sometimes dead zones are operating conditions that the algorithms of control are not design for. The supervisor takes these situations into account, in order to maintain the stability and robustness of the systems (Hägglund & Aström, 2000).

The problem is the control of complex systems for traditional methods of control to provide a reasonable performance. There are several ways to improve the performance of a system applying the supervision to control methods, such as: switching control, reconfigurable control and adaptive control.

The main objective of the supervisor is to monitor the signals that can be measured and decide, in real time, whether to apply a different controller or just adapt the controller's parameters (J. Hespanha, 2001).

In Figure 2.19 the components of a supervisor are embodied. $\mathbb{P}$ represents a process, $\mathbb{C}$ a controller, $\mathbb{E}$ an estimator, $\mathbb{M}$ the monitoring signals and $\mathbb{S}$ signifies the switching logic (J. Hespanha, 2001).

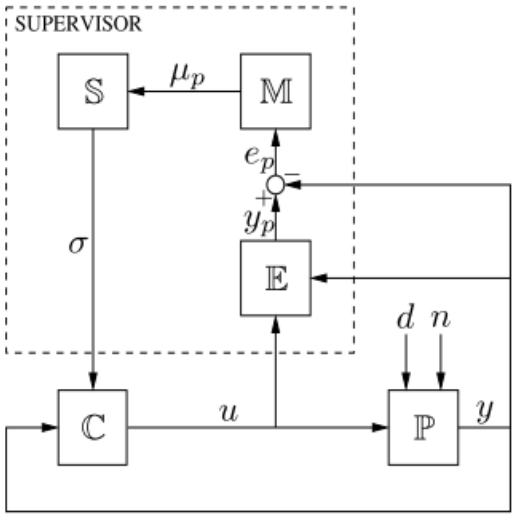


Figure 2.19 - Generic supervisory diagram (J. Hespanha, 2001).

Supervised control systems tend to appear in real life application where faults are a concern. Because of this, fault detection/diagnosis and fault tolerant control are commonly seen side by side with the supervisor. More on Fault detection/diagnosis and fault tolerant control will be exploited in the next topic.

2.3.8 Fault Detection/Diagnosis and Fault Tolerant Control

In the generic common sense, a failure is something that changes the behavior of a system in a way that it no longer satisfies its purpose. It may be an internal event in the system breaking the power supply, information links or for instance, leakages in pipes (Blanke *et al*., 2006). Faults are a deviation from the accepted behavior, of a property of a system, they can lead the system to a failure (Cardoso, 2006).

In order to better understand these events, plant faults can be categorized in three groups: actuator faults, component faults (faults in the framework of the process) and sensor faults (Frank, 1996). Figure 2.20 gives a diagram of these categories. The propagation of faults can deteriorate or damage machines and humans; therefore, the quick discovery of faults is imperative in order to stop the propagation of their effects. The controller should be able to measure these fault effects and make the system fault tolerant. In case of success, the system may be able to satisfy its purpose, possibly after a short time of degraded operation (Blanke *et al*., 2006).

The faults can be described as input signals. The modeling uncertainty, because of the un-modeled disturbances, noise and model mismatch, may not be critical to a system operation but can raise false alarms which respect the fault detection. Faults can also take place in other systems such as controllers or supervision (Brito Palma, 2007).

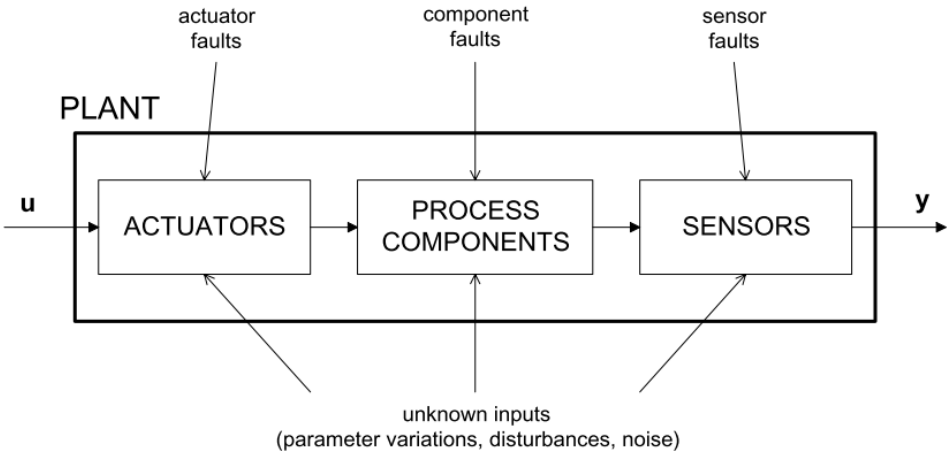


Figure 2.20 - Categories of faults in a system (Brito Palma, 2007).

It is very important to note the distinction of notions between fault and failure. As explained *supra,* faults cause a change in the characteristics of a component, changing the component performance and operation mode in an undesired way. Still, a fault can be dealt with by a fault tolerant control, keeping the faulty system operational (Cardoso, 2006).

On the other hand, failure is characterized by incapacitating the system or component to fulfill its function. Failure, as it is an irrecoverable event, implies the shutdown of the system. With these notions clear, the idea of fault tolerant control is to prevent a fault from causing a failure at the system level (Blanke *et al*., 2006).

There are certain requirements and properties which respect systems subjected to faults:

- Safety is a major requirement in order to protect technology from permanent damage as well as humans. The inability to shut-down immediately and guarantee the system to reach a safe status is also a requirement (Blanke *et al*., 2006).
- Reliability is the probability that a system can fulfill its purpose for a specific period of time, under normal condition. Fault tolerant control have no influence in the reliability of the process components; although it has a major influence in the overall system reliability (Cardoso, 2006).
- Availability is the likelihood of a system to be operational when needed. This systems' property depends on the maintenance policies (Cardoso, 2006).

- Dependability includes all the three properties *supra*. A dependable system is a fail-safe system with great reliability and availability (Brito Palma, 2007).

Because safety is the most important of all four properties mentioned before, its relation with fault tolerance is now explained in more detail. Assuming that the system's performance can be described, for instance, by a two dimensional space.

Figure 2.21 shows the two dimensions where the performance regions are delimited. In the region of required performance, the system fulfills its function. This is where the system should be during its operation time. The controller assures that the system remains in this region despite disturbances or model uncertainties used in the controller design

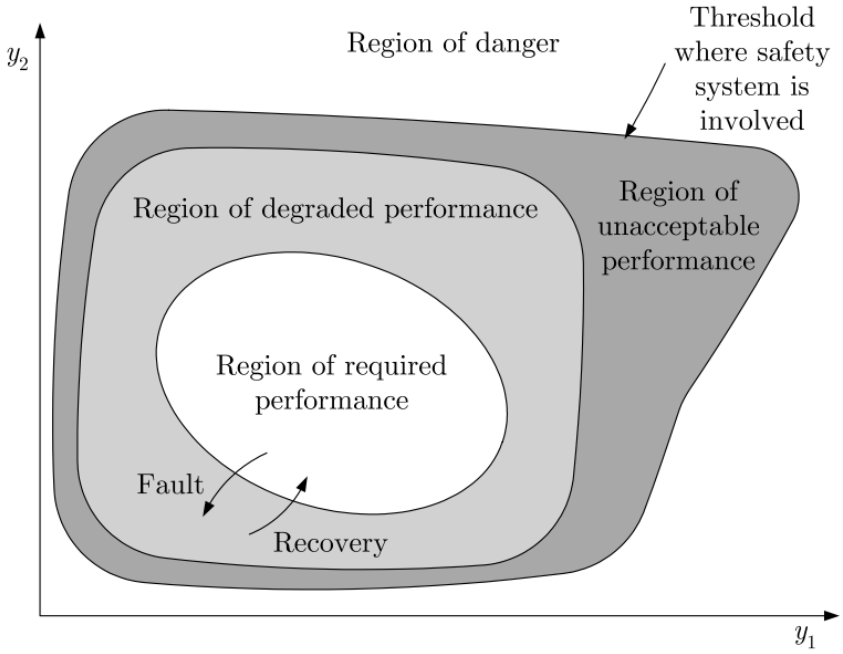


Figure 2.21 – Two dimensional performance regions (Blanke *et al*., 2006).

The region of degraded performance demonstrates where the faulty system is allowed to stay. Faults are the reason for the system to go from required performance to degraded performance. Although the performance in this region is not the required, the system can still operate with considerably degraded performance. Fault tolerant controllers should be able to bring the system back from degraded performance into required performance in order to prevent the system from falling into unacceptable or dangerous performance. At the borders the supervision system enters in action, diagnosing the fault and adjusting the controller to the new situation (Blanke *et al*., 2006).

The region of unacceptable performance should never be reached - it lies between the regions of acceptable performance and the danger zone where the safety of the system is on the line. In order to stay out of the danger zone, the system interrupts its operation avoiding damage to itself and its surroundings. This shows that safety systems and fault tolerant controllers work in separate regions allowing its design without the need to meet safety standards (Blanke *et al*., 2006).

Before jumping into fault tolerant control architectures, fault types regarding temporal behavior must be defined. They can be defined in additive, multiplicative or intermittent which are represented in Figure 2.22 (Cardoso, 2006).

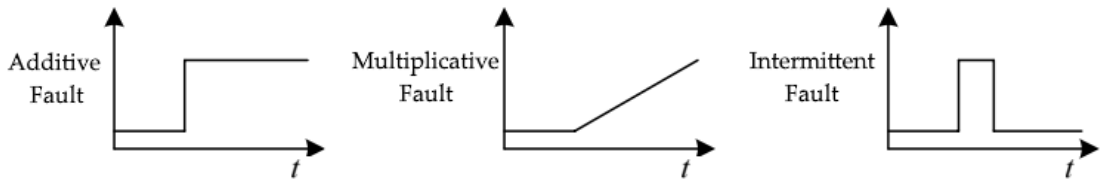


Figure 2.22 - Fault types regarding temporal behavior (adapted from Cardoso, 2006))

Regarding structures of fault tolerant controllers, there are several possibilities. One example of a fault tolerant control structure is represented in Figure 2.23. There are two major blocks regarding the fault tolerant control:

- Fault diagnosis: where the existence of faults must be detected and identified. This is done by measuring inputs and outputs and testing their consistency with the model (Brito Palma, 2007).
- Control re-design: has the goal of adapting the system to the fault thus allowing it to continue its objective. This is accomplished by using the information from the diagnosis block (Brito Palma, 2007).

These components cannot be done by an usual feedback controller; instead they need a supervision system (2.3.7 Supervised Control), in order to advocate the control scheme and select the algorithm and/or parameters of the feedback controller (Blanke *et al*., 2006).

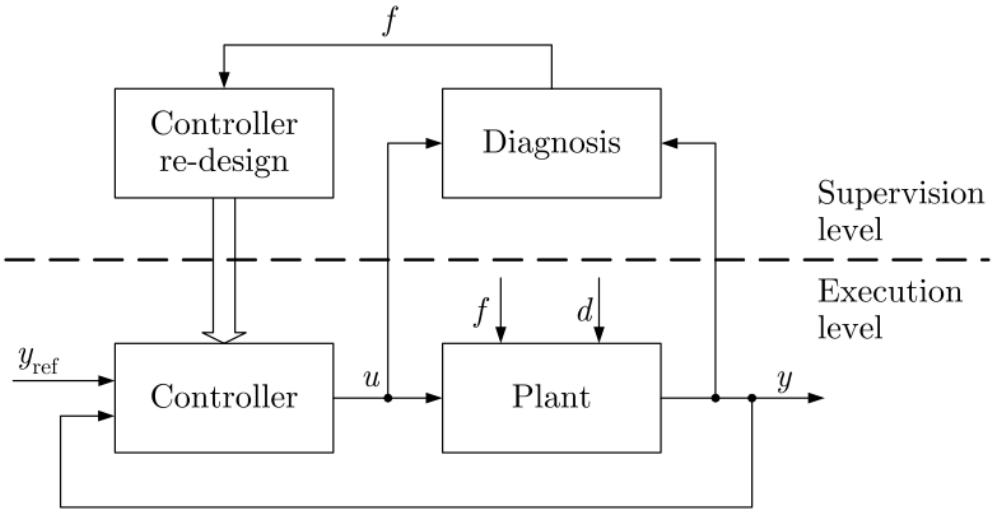


Figure 2.23 - Fault tolerant control architecture (Blanke *et al*., 2006).

There are three methods used to stablish a fault tolerant control: Reconfigurable Control, Robust Control and Adaptive Control. Adaptive Control and Reconfigurable Control are considered active control technics in response to faults. Differently, Robust Control is a passive method of reacting to faults.

2.3.9 Reconfigurable Control

In a reconfigurable control system, control allocator (2.3.11 Control Allocation) algorithms are needed to perform the distribution of control, while keeping the system stable, obeying to the rate and position limits of actuators and hypothetically recovering from non-nominal conditions (Oppenheimer, Doman, & Bolender, 2006).

This approach is commonly the second step of fault tolerant control, dealing with the possibility of an important signal path (actuator, sensor, etc.) being broken. In order to restore the control of the system, it is required to find a new solution for the system in order to circumvent the broken link. As the signal path is part of the physical design and construction of the system, it is almost never changeable or repairable right away. However, changes can be made to the control structure of the system (Steffen, 2005). Figure 2.24 shows a diagram of reconfiguration of control reacting to either a broken sensor or an actuator.

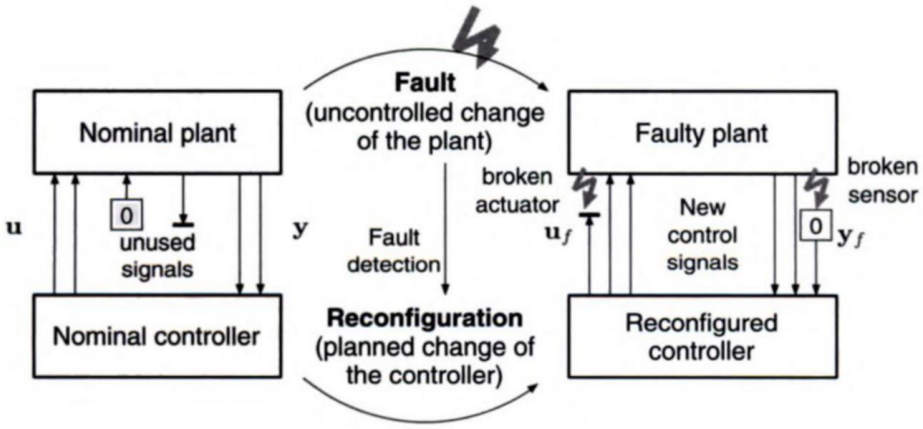


Figure 2.24 - Reconfiguration control in response to a fault (Steffen, 2005).

2.3.10 Over-actuated Systems

Recently, much importance has been placed on over-actuated systems. They can provide redundancy for systems, allowing potential recovers from non-nominal conditions. When a system has more actuators than axes to control, it is considered an over-actuated or redundant system. Because of this redundancy, Control Allocation, *infra*, is commonly used in over-actuated systems. The allocation or mixing of these control actuators to reach a desired objective is the control allocation problem (Oppenheimer *et al*., 2006).

2.3.11 Control Allocation

Control allocation algorithms are used in automatic distribution of control in a system. There are several techniques to allocate control such as explicit ganging, pseudo control, pseudo inverse and daisy chaining. As these are the simplest methods, they have disadvantages – neither can guarantee the violation of the rate and position limits of actuators. Some of them are even difficult to apply due to the need to derive a control mixing law *a priori.* Control allocation algorithms are used in order to compute a unique solution to the problem.

Usually, control allocation algorithms are seen as optimization problems, making all degrees of freedom available to use and, if necessary, allowing secondary objectives to be achieved if needed. (Oppenheimer *et al*., 2006). There is another control allocator method called direct allocation. This one finds the control vector that results in the best estimation of the command vector (Durham, 1993). The methods mentioned before are well explored in Oppenheimer *et al*., 2006.

2.3.12 Optimal Control

Optimal control deals with the problem of discovering the control parameters for a certain system, in order to fulfill a certain criteria, usually called cost function. The technique emerged in the 1950s from the work of Lev Pontryagin and Richard Bellman (Bryson, 1996).

An optimal control is composed by a set of differential equations that describe the path of control variables that minimize the cost function. There are many optimization approaches to solve engineering and control problems, such as neural networks and genetic algorithm (Brito Palma, Vieira Coito, Gomes Ferreira, & Sousa Gil, 2015). The Particle Swarm Optimization is amongst them and will be described *infra.*

2.3.12.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimization algorithm that proved to be an efficient way to discover controllers' parameters. From PID controllers to more complex control architectures, such as SMC controllers, the PSO is able to optimize the control parameters, in order to minimize the cost function. The algorithm described is based on the work of Kennedy & Eberhart, 1995.

Each particle is characterized by their position and speed on a controller's parameter space being updated according to equation (2.3.10).

$$p_i(n) = p_i(n-1) + s_i(n) \tag{2.3.10}$$

The position of the particles i at iteration n is $p_i(n)$ and $s_i(n)$ is the speed. The speed is updated according equation (2.3.11).

$$\begin{aligned} s_i(n) = w\, s_i(n-1) \\ + c_1\, rand(.)\, \big(o_i(n-1) - p_i(n-1)\big) \\ + c_2\, rand(.)\, \Big(o_g(n-1) - p_i(n-1)\Big) \end{aligned} \tag{2.3.11}$$

Where w, c_1 and c_2 are real value parameters, $o_i(n)$ is the location of the best position of particle i history, $o_g(n)$ is the location of the best position found by any particle and the random real value vector *rand(.)* is homogeneously distributed in the range [0; 1]. Each particle position is evaluated using a cost function.

2.4 Related Work

Along the years, there has been some developments in multirotors research area. Up next there are some recent works with some knowledge bases for the research of this thesis.

Although there are earlier developments in the multirotor recent history, let's start in 2007, with the work done in *École Polytechnique Fédérale de Lausanne* by Samir Bouabdallah. This thesis concerns the design and modelation of a miniature flying robot, in one word, a micro-quadcopter. The focus was the vertical take-off and landing. There were several control techniques used in this work, such as control based on Lyapunov theory, PID and linear quadratic and, last but not least, backstepping and sliding-mode methods (Bouabdallah, 2007). The prototype developed in this thesis is in Figure 2.25.

Figure 2.25 - Samir Bouabdallah's micro-quadcopter (Bouabdallah, 2007).

In 2008 the *Instituto Superior Técnico da Universidade Técnica de Lisboa* published a master's thesis in the quadcopter area. The work was done by Sérgio Eduardo Aurélio Pereira da Costa with the title "Controlo e Simulação de um Quadrirotor convencional". The work had the objective of modeling, simulating and controlling an unmanned aerial vehicle with rotary thrusters. The design was similar to a quadcopter containing four rotors and a X shaped frame. Pereira da Costa implemented a Kalman filter, in order to observe the global sates of the system, a linear quadratic regulator and a linear Gaussian regulator (Pereira da Costa, 2008). The prototype is in Figure 2.26.

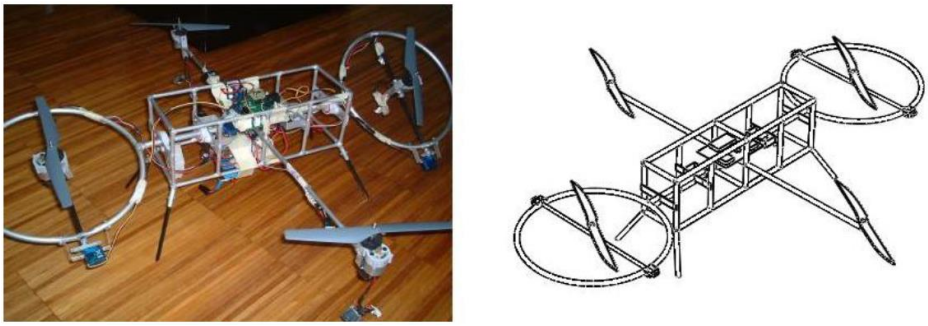


Figure 2.26 – Sérgio Costa's prototype (Pereira da Costa, 2008).

Later, in 2011, there was a very complete project in *Faculdade de Engenharia da Universidade do Porto* by José Duarte Alves de Sousa. His work was to design a dynamic model and simulation of an autonomous four rotor aerial vehicle. He applied three different control techniques: Fuzzy, PD and PID. Afterwards, he compared their performances. He also developed an user interface in LabVIEW, allowing to control the vehicle (Alves de Sousa, 2011). His prototype is in Figure 2.27.

Figure 2.27 - José Sousa's Quadcopter (Alves de Sousa, 2011).

More recently, in 2012, there was a paper published in *The International Journal of Robotics Research* where the study intent was to design dynamically feasible trajectories and controllers. The paper objective was the development of trajectories and controllers that allow the conventional quadcopter to fly aggressively, maneuvering through narrow spaces, vertical gaps, and land in any angle surface with high precision. The controllers were designed according to a dynamic model and then adjusted in an automated method to the real quadcopter Figure 2.28. Experimental trials took place and are available for watching at https://www.youtube.com/watch?v=MvRTALJp8DM (Mellinger, Michael, & Kumar, 2012).

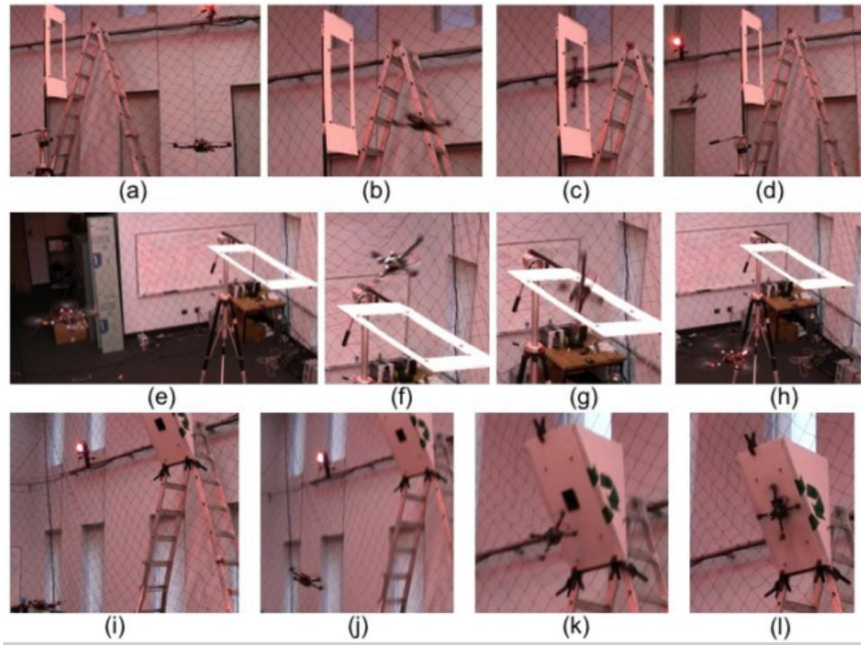


Figure 2.28 - Images from experimental trials. (a)-(d) Passing through a vertical window. (e)-(h) Passing through a horizontal window. (i)-(l) Landing in a 120º angle.

Besides these published works there are some industrial application already on the move like the DJI Phantom 4 (DJI, 2016) for cinematography, Amazon's PrimeAir (Amazon Inc, 2016) for package delivery, precision farming, inspection (Flyablility SA, 2014), emergency services (Momont, 2014) among other applications.

Chapter 3

3 Modeling, Identification and Control

3.1 Introduction

This section introduces the X8-VB Quadcopter. First, there is a detailed description of the hardware involved in the structure from the power source to the processing unit. Modelling and control are next, where the model techniques and parameters are explored and then control algorithms are developed. Last but not least, fault tolerance control algorithms take place. This chapter follows the diagram already mentioned in 1. Introduction and also presented next in Figure 3.1.

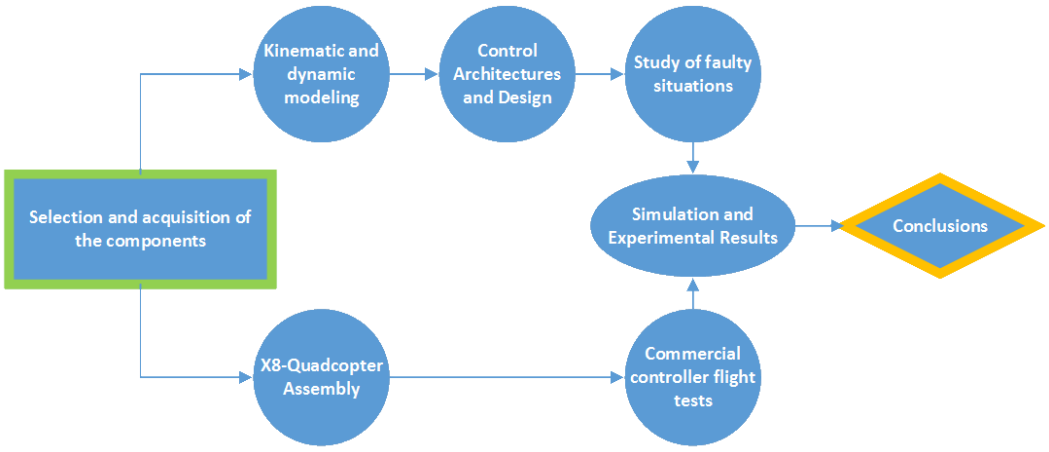


Figure 3.1 - Workflow diagram.

3.2 High Level Architecture

The high level architecture can be seen in Figure 3.2. A specific explanation of each block will be described next. The specifications of each component is presented in order to introduce their main characteristics and the ones that affect the aircraft directly.

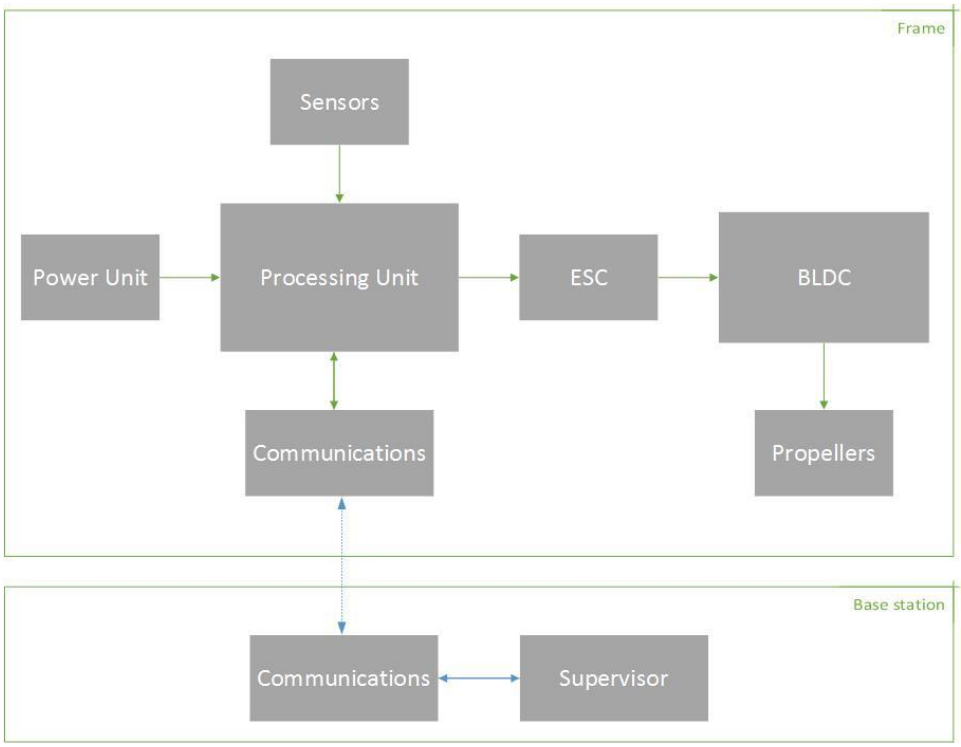


Figure 3.2 - High level hardware architecture.

3.3 Hardware Architecture and Specifications

3.3.1 Frame

The frame was chosen according to the size and material resistance, since the possibility of malfunctions and falls are to be considered, as this is only a prototype where experimental testing will be implemented. DJI Flamewheel 450 was chosen due to resistance and building quality.

The choice of this frame takes into account the fact that it is not prepared to assemble eight motors and, because of this, there was the need to build an adapter. In order to do this, a 3D printer and Autodesk123D were essential to create and then print the adapter.

The adapter was made out of plastic and contemplates two parts - the upper and lower one, as shown in the Figure 3.3.

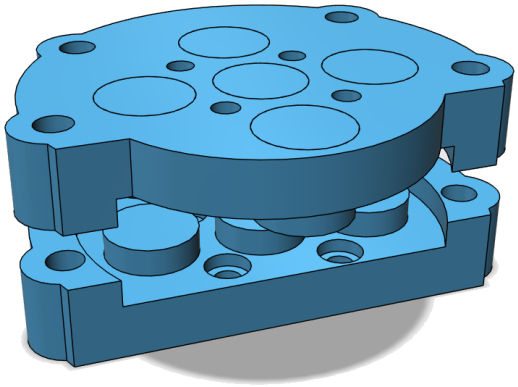


Figure 3.3 – 3D model of upper and lower sides of the adapter.

After assembling the eight motors with the propellers, there was the need to build another extension in order to make space for the propeller to rotate freely. With this extension, it was possible to create another level for the battery lowering the center of gravity. The extension is shown in Figure 3.4.

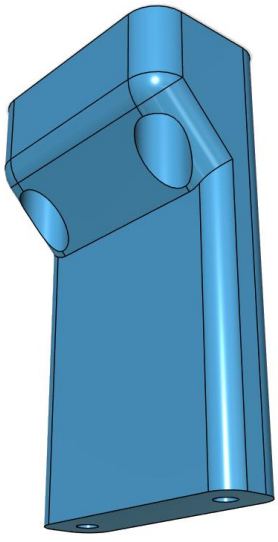


Figure 3.4 – 3D model of the extension piece.

The creations of the complete 3D model (Figure 3.5) then took place in order to implement a virtual simulation. A problem was encountered at this stage. Because the model was too rigorous, Simulink 3D was not able to work with it due to the heavy computational calculations. The solution was to create a simplified model (Figure 3.6) just to represent the process dynamics in the virtual world.

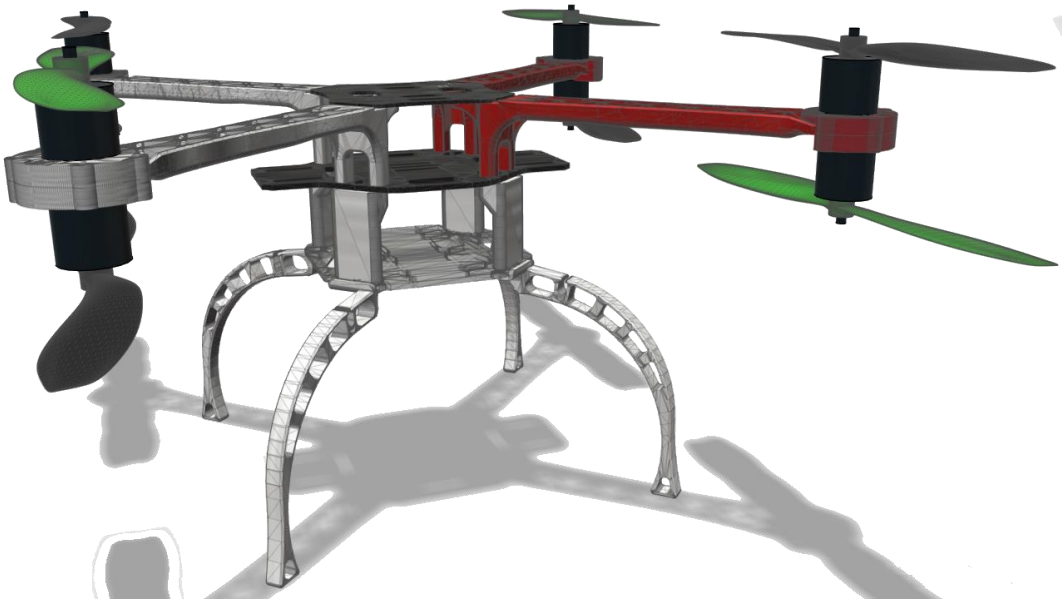


Figure 3.5 - Complete 3D model.

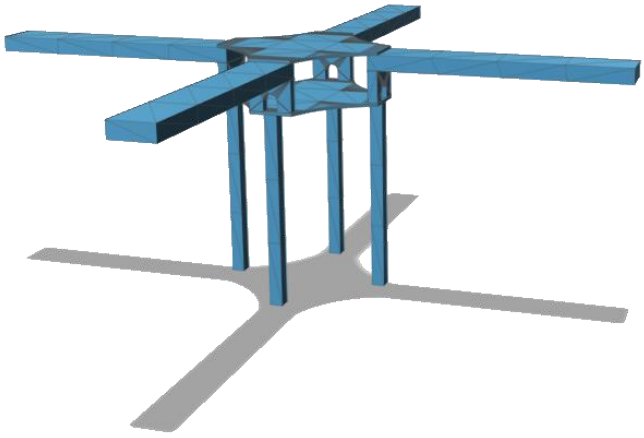


Figure 3.6 - Simplified 3D model.

3.3.2 Power Unit

As previously mentioned, LiPo batteries are the most common power supplies among drones. Due to the discharge rate needed to the BLDC motors, the LiPo technology is the best choice. Another very important property of these types of batteries is the power density, demonstrated in Figure 3.7.

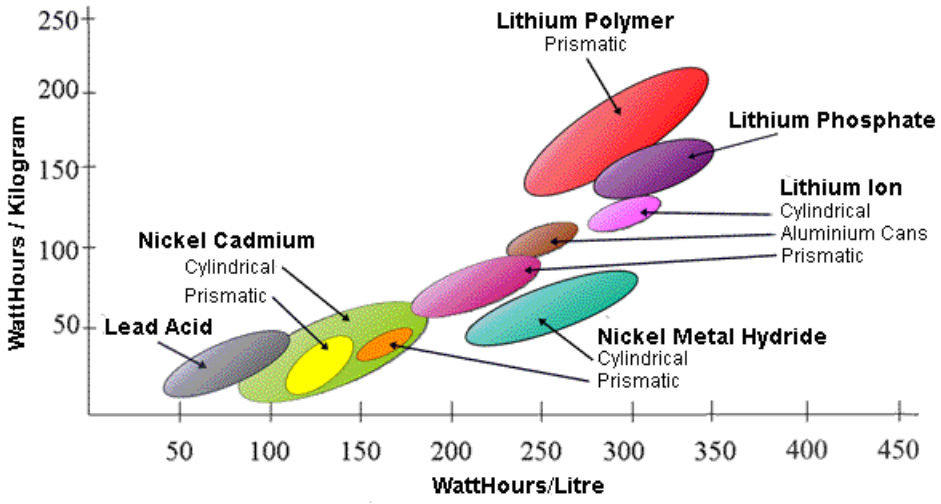


Figure 3.7 - Weight and dimension related to power density[3] (Alves de Sousa, 2011).

Two types of batteries were chosen in this research: the first with 5000 mAh of capacity and 25-35C of discharge rate and the second with 6000 mAh of capacity and 25-50C of discharge rate. The different capacity size and discharge rate were an approach to study the difference of performance and fly time of the multirotor. Both the batteries have 4 Cells with sum to 14.8 V of output voltage. They also have matched impedance. Figure 3.8 shows both selected batteries.

[3] http://www.mpoweruk.com/chemistries.htm

Figure 3.8 - On the left, the 6000 mAh battery. On the right, the 5000 mAh battery.

A very important note is that each cell inside the batteries should never be lower than 3V. In case of voltage lower than 3V, the damage to the battery cells could be destructive or even ignite the battery.

3.3.3 Processing Unit

In order to control the multirotor, there will be two controllers on board. One is the result of the controllers designed in this research and the other one works as a failsafe, in case there is a misplaced controller or a controller fault. The latter is a NAZA Lite V2 from DJI, which will only be activated in case of emergency, applying its control only to the upper motors. This presents the tolerant control even if the controller design in this work fails. The switching command is operated manually with the human interface controller.

The controllers designed will be implemented in one Arduino DUE. This board contains a 32-bit ARM core microcontroller working at 84 Mhz, along with 54 digital input/output pins (of which 12 can be PWM outputs) and 12 analog inputs. A more complete description can be seen in Table 3.1.

Table 3.1 - Arduino DUE specifications (Source: Arduino).

Specification	***Value***
Microcontroller	AT91SAM3X8E
Operating Voltage	3,3 V
Input voltage (recommended)	7 – 12 V
Input voltage limits	6 – 16 V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12

Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

3.3.4 Communication Modules

In order to communicate with a base station and with the human controller, there are two communication lines - 433 Mhz radio frequency (RF) and 2.4 Ghz Wi-Fi frequency. The two frequencies were necessary in order to prevent miss communication or interference between commands.

3.3.4.1 RF Communication

The 433 Mhz radios are used to allow the on-board data to reach the base station. The radios are a 2 way full-duplex through adaptive TDM, making it possible to send and receive data simultaneously, with the ability to use “clear to send” and “request to send” signals (see Figure 3.9) The serial communication is made at a 57600 baud rate - in other words, symbols per second or pulses per second.

Figure 3.9 - On the left the pin-out descriptions of the radio. On the right an illustration the receivers.

The connection to the base station are made by USB processed by Matlab®. The connections to the Arduino DUE are represented in Figure 3.10, using the serial communication (RX/TX).

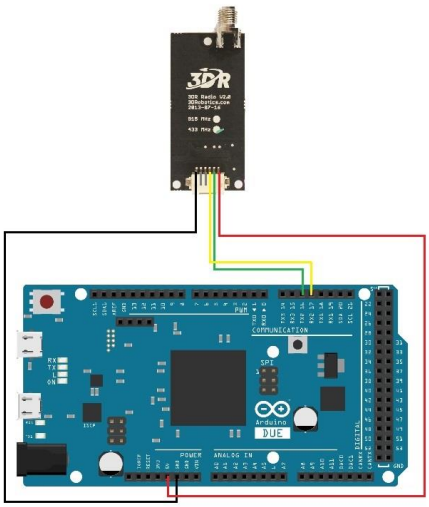


Figure 3.10 - Arduino Due and 3DR pin connections.

The 3DR Radio V2 is based on HopeRF's HM-TRP module as it is equipped with UART interface, transparent serial link, and MAVLink protocol framing. It is also capable of reconfiguring the duty cycle and error correction up to 25% of the bit errors. More detailed specifications are presented in Table 3.2:

Table 3.2 - 3DR Radio V2 specifications.

Specification	***Value***
Max. Output power	100 mW
Receive sensitivity	-117 dBm
Operation frequency	433 – 915 mHz
Supply voltage	3.7 – 6 Vdc
Transmit current	100 mA at 30 dBm
Receive current	25 mA
Serial interface	3.3 V UART
Dimensions	26.7×55.5×13.3 cm

3.3.4.2 Human Interface Controller

The controller is composed by two elements: the controller itself and the receiver. The controller is where the human interface is and the receiver is assembled in the aircraft, in order to receive the commands in real time.

The controller chosen was the FrSky Taranis X9D Plus (Figure 3.11). The choice was not only because it runs over an open source system but also because of its exceptional programed communication protocol. The controller provides innumerous buttons in order to interact with the aircraft. One of the buttons is particularly important because it allows the change between control units.

Figure 3.11 – FrSky Taranis X9D Plus on the right and the receiver on the left.

The receiver FrSky X8R (Figure 3.11) has 8 channel outputs and supports telemetry system; therefore, some flight parameters are shown in the controller's screen in real time. The parameters are: the altitude, the GPS coordinates and the battery voltage. The receiver, beyond the smart port where the sensors are connecter, also supports SBus and RSSI ports.

3.3.5 Sensors

There are 3 different kinds of sensors involved in the aircraft system. The sensors connected to the Arduino DUE, the ones connected to the NAZA Lite and the sensors connected to the FrSky X8R. A brief explanation and specifications of the sensors connected to the Arduino DUE is presented up next. The other sensors are part of a product and have the same working principle.

3.3.5.1 GPS

The GPS sensor is the GTOP PA6B and is integrated in a shield that also includes a Micro SD card for data logging. The GPS comes along with an integrated ceramic antenna and a Farad capacitor to replace the external battery for real time clock (RTC) use. Table 3.3 shows some specifications:

Table 3.3 - Shield GPS Logger specifications

Specification	*Value*
Sensibility	-165 dBm
Frequency of update	10 Hz
Channels	66
Current consumption	20 mA

The shield can be seen in Figure 3.12.

Figure 3.12 - Shield GPS Logger.

3.3.5.2 Absolute Orientation Sensor

The Absolute Orientation Sensor is a 9 axis motion shield from Bosch. The shield is based on the BNO055 orientation sensor. It has an integrated triaxial 14-bits accelerometer, a triaxial 16-bits gyroscope with ±2000 degrees per second, a triaxial geomagnetic sensor and a 32-bits microcontroller running the BSX3.0 FusionLib software.

The signals provided by the shield are:

- Quaternion;
- Euler angles;
- Rotation vector;
- Linear acceleration;

- Gravity vector.

The consumption specification are shown in Table 3.4.

Table 3.4 - 9-Axis shield power specifications.

Specification	***Value***
Operating voltage	5 V
Power consumption	50 mW

The shield is represented in Figure 3.13.

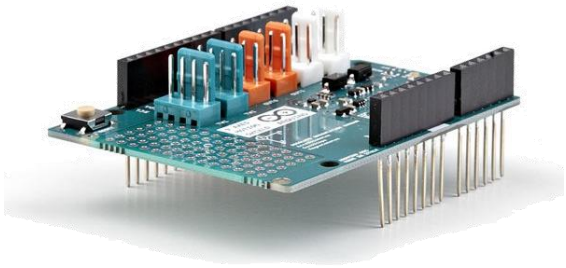


Figure 3.13 - 9-Axis Shield.

3.3.5.3 AltiMU

The AltiMU sensor will work alongside with the 9-Axis shield, allowing sensorial fusion, in order to obtain better information of the ongoing flight. The AltiMu sensor is equipped with an integrated triaxial 16-bits accelerometer, a triaxial 16-bits gyroscope, a triaxial 16-bits magnetometer and a 24-bits barometer.

The consumption details are shown in Table 3.5

Table 3.5 - AltiMU power specifications.

Specification	***Value***
Operating voltage	2.5 – 5.5 V
Power consumption	15 - 33 mW

The sensor is represented in Figure 3.14.

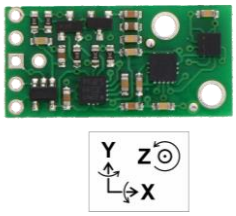


Figure 3.14 – AltiMU sensor.

3.3.5.4 ACS712 Current Sensor

This current sensor (Figure 3.15) was used to detect malfunctions of the motors, with the range of -30 to 30 A and connected to the analog port of the Arduino. Some specifications are presented in Table 3.6.

Figure 3.15 -ACS712 Current Sensor.

Table 3.6 - Current sensor specifications.

Specification	***Value***
Operating voltage	5 V
Analog output	66 mV/A
Current input limit	-30 – 30 A
Sampling time	5 μs

3.3.6 Electronic Speed Controllers

The Electronic Speed Controllers (ESC) were chosen according to the needs of the BLDC motors. The model chosen was the 420 Lite from the E Series, with square-wave architecture. The ESC is in Figure 3.16, where we can see the power and the command cables, as well as the three phase connectors for the BLDC motors.

Figure 3.16 - ESC 420 Lite.

More information is in Table 3.7.

Table 3.7 - ESC specifications.

Specification	***Value***
Max. Allowable Voltage	17,4 V
Max. Allowable Current (Persistence)	20 A
Max. Allowable Peak Current (3 seconds)	30 A
PWM input signal level	3,3 V / 5 V
Signal Frequency	30 ~ 450 Hz
Battery	3S ~ 4S LiPo
Size	54×24×9 mm
Weight	27 g

The PWM range of input is from 1150 to 1850 μs of uptime.

3.3.7 Brushless DC motors

The E305 2312E BLDC (Figure 3.17) motors were selected, due to its efficiency and robustness. Their optimized electro-magnetic system boosts the efficiency by around 7%, when compared to the previous series. The lower cogging torque provides a smoother performance at low RPMs.

Figure 3.17 - BLDC motor

There are eight of these motors in the quadcopter - four working alternately clock-wise (CW) and counter-clock-wise (CCW) on one side and the another four motors working reversely on the down side. More on how they interact with the aircraft system in section 3.4.1.

The specifications can be found in Table 3.8.

Table 3.8 - BLDC specifications

Specification	***Value***
Max. Thrust	850 g/rotor
Rotation	960 rpm/V
Weight	57 g
Stator Size	23×12 mm

3.3.8 Propellers

The propellers were chosen by taking into account that they needed to be assembled from both sides. There are eight propellers in the multirotor - four working in CW rotation and the other four working in CCW rotation. The propeller is 100 mm long with a pitch of 4.5 mm, as we can see in Figure 3.18. X and Y are the length and the pitch, correspondently.

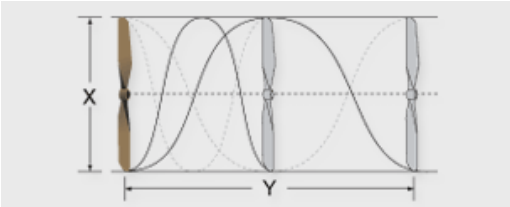


Figure 3.18 - Propeller rotation plot

In Figure 3.19 are the CCW propellers.

Figure 3.19 - CCW propeller

3.4 Multicopter Modeling

The objectives regarding the model are to reliably reproduce the aircraft's dynamic and kinematic, in order to posteriorly develop and apply fault tolerant control algorithms.

3.4.1 Specifications and Operation Principles

The Quadcopter X8-VB is shaped as a cross, with its arms having a 90º angle between each other. The thrusters are assembled at the end of the arms with propellers assembled on them - they are responsible for the horizontal and vertical forces.

The control of the aircraft is directly related to the motors and the propellers, since these are the elements that interact with the maneuvering of the frame and the speed of the maneuvers.

3.4.1.1 Altitude (U_1 [N])

In order to manipulate the altitude, the movement in relation to the Z_E axis, all the thrusters should be rotating at the same speed, generating equal force. The sum of all eight forces that take action in the altitude manipulation is represented as U_1. The mathematical formulation of this force is represented in equation (3.4.1).

$$U_1 = \sum_{i=1}^{8} f_i \tag{3.4.1}$$

The physical representation of U_1 can be seen in Figure 3.20.

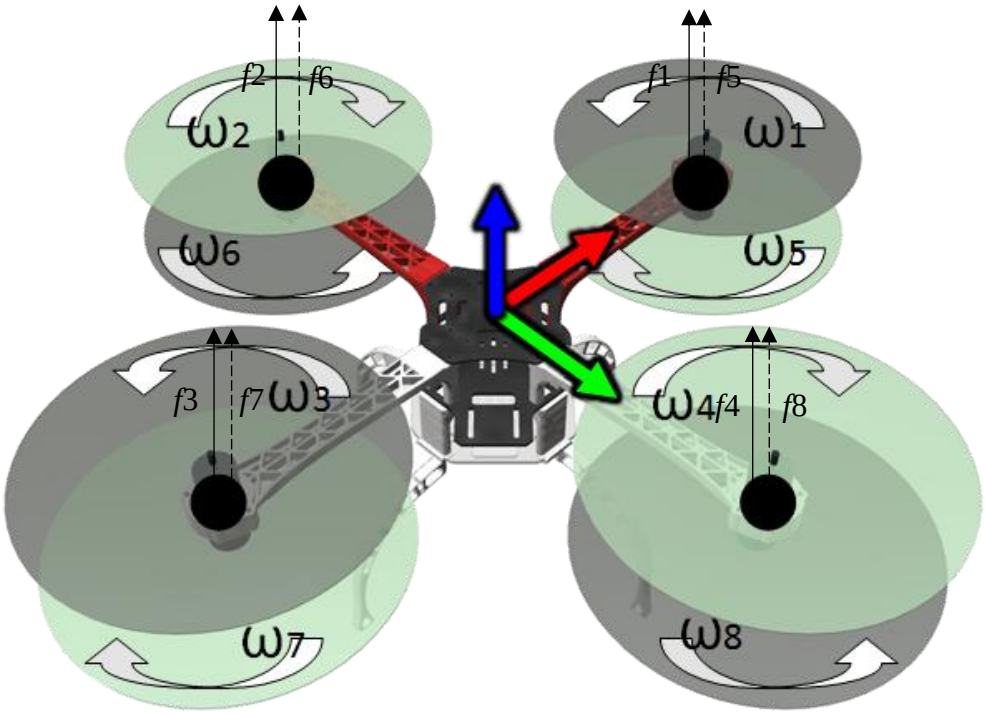


Figure 3.20 - Representation of equal rotations and forces. RGB arrows correspond to XYZ axis respectively.

Figure 3.20 shows forces (f_n [N]) of each thruster and rotation (ω [rad/s]). The filled black arrows correspond to the upper thrusters, and the dashed ones to the lower side thrusters.

In order to move the drone up and down there are two important rules:

- To increase altitude, increasing the value of the Z axis, the sum of force U_1 must be greater than the gravity force P of the multirotor.

$$U_1 > \mathrm{P} \tag{3.4.2}$$

- To decrease altitude, decrementing the value of the Z axis, the sum of force U_1 must be less than the gravity force P of the multirotor.

$$U_1 < \mathrm{P} \tag{3.4.3}$$

3.4.1.2 Roll ϕ (U_2 [N])

Roll is the rotation around the X axis. The rotation is measured in radians. To make it more perceptible, a conversion to degrees is made in the plots. In order to drag the aircraft left or right, the sum of forces has to vary accordingly. This rotation potentiates a displacement in relation to the Y axis. U_2 represents the sum of forces involved in the roll rotation.

$$U_2 = f4 + f8 - f2 - f6 \tag{3.4.4}$$

Equation (3.4.4) represent the positive roll rotation forces distribution. The forces not mentioned in the equation must maintain their force in order to keep the stability. An example is in Figure 3.21.

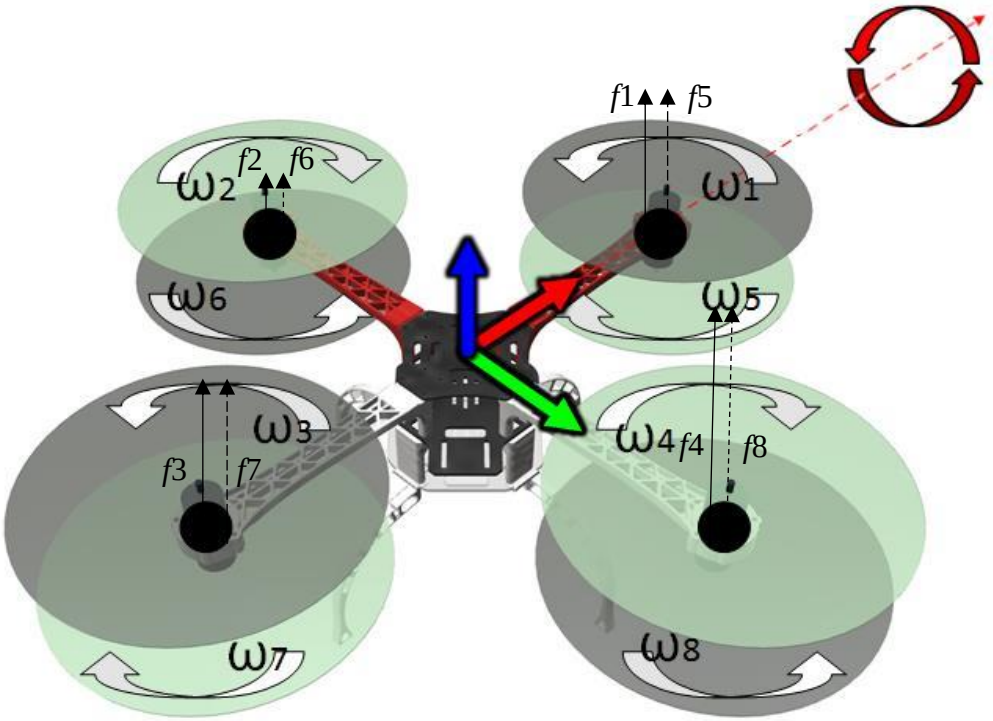


Figure 3.21 - Representation of a positive roll rotation forces.

3.4.1.3 Pitch θ (U_3 [N])

Pitch is the rotation around the Y axis. As the roll, the rotation is measured in radians and, to make it more perceptible, a conversion to degrees is made. In order to drag the aircraft front or backwards, the sum of forces produced by the thrusters must diverge accordingly. This rotation potentiates a displacement in relation to the X axis. U_3 represents the sum of forces involved in the pitch rotation.

$$U_3 = f3 + f7 - f1 - f5 \tag{3.4.5}$$

Equation (3.4.5) represent the positive pitch rotation forces distribution. The forces not mentioned in the equation must maintain their force in order to keep the stability. This example is in Figure 3.22.

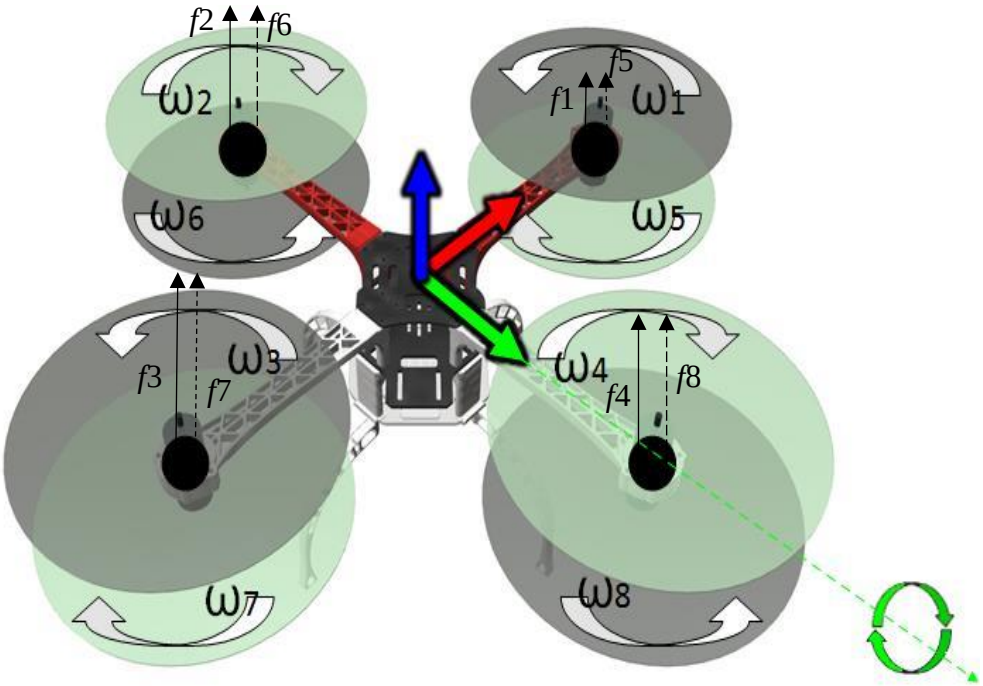


Figure 3.22 - Representation of a positive pitch rotation forces.

3.4.1.4 Yaw ψ (U_4 [N])

This angle is associated with the orientation of the aircraft, that is a rotation around the Z axis. As the maneuvers mentioned, the angles are calculated in radians and then converted into degrees. The rotation is made in CCW direction. In order to operate a change in the orientation, all eight thrusters must be used. U_4 represents the sum of forces involved in the yaw rotation.

$$U_4 = f1 + f3 + f6 + f8 - f2 - f4 - f5 - f7 \tag{3.4.6}$$

Equation (3.6.1) represent the positive yaw rotation forces distribution. Figure 3.23. is an example.

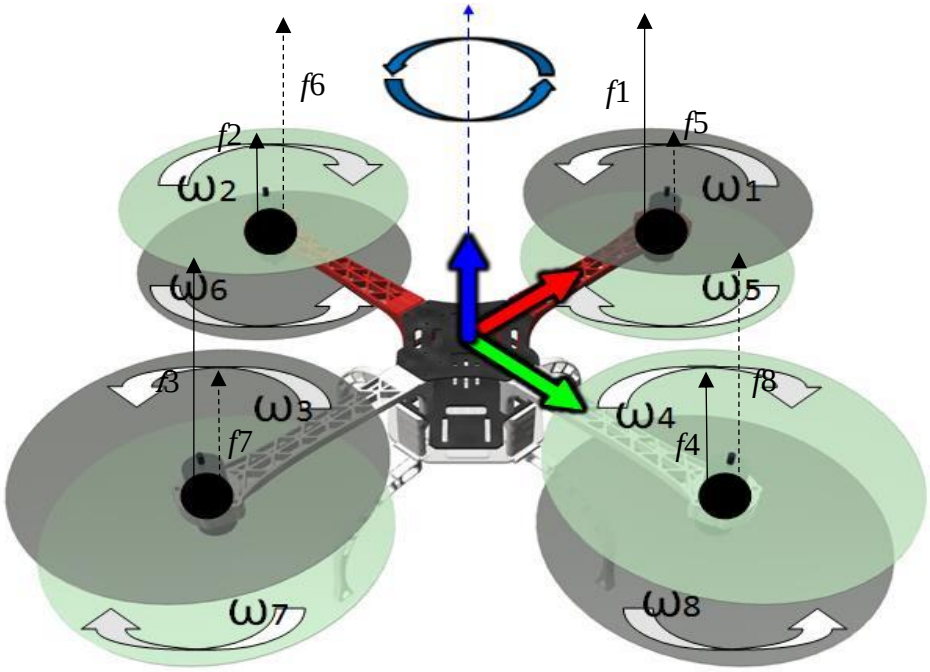


Figure 3.23 - Representation of a positive yaw rotation forces.

3.4.2 Kinematic and Dynamic Model

In order to develop the model of the aircraft, some assumptions took place. We assumed that the frame and its components were a rigid and symmetric structure. The referential was placed in the structure at the center of mass, and the axis of the referential were placed overlapping the inertial axis of the multirotor, as the previous section figures demonstrated. The forces associated to the thrusters are proportional to the square of their speed.

Beyond the referential fixed in the aircraft (X_A, Y_A, Z_A), to obtain the model equations, one must place another fixed referential at a reference point. In our case, the second referential was placed on earth (X_E, Y_E, Z_E). This referential is fixed in relation to the aircraft that moves according the multirotor movement.

The earth referential is used to obtain the angular $\boldsymbol{\Theta}_E$ [rad] and linear position $\mathbf{P}_E$ [m] of the aircraft in relation to the earth. The aircraft referential is used to obtain the linear ($\boldsymbol{\mathcal{V}}_A$ [m·s^{-1}]) and angular ($\boldsymbol{\Omega}_A$ [rad·s^{-1}]) velocity, the forces $\mathbf{F}_A$ [N] and the torque $\mathbf{T}_A$ [N·m].

The linear position $\mathbf{P}_E$ is determined by the vector originated from the union of the origins of the two referentials, as shown in Figure 3.24.

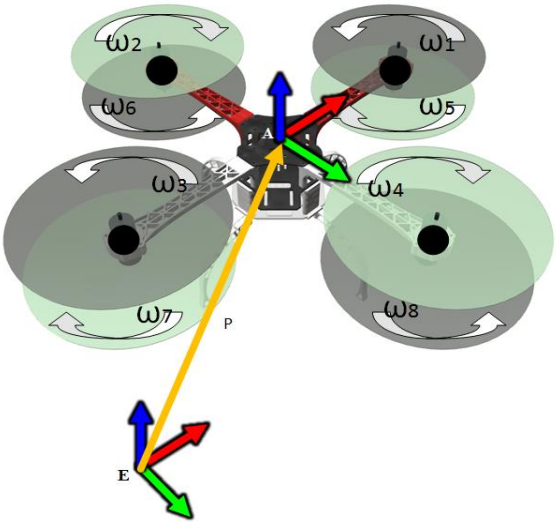


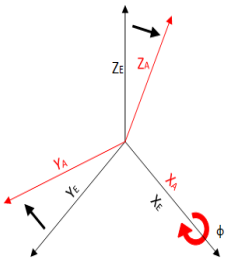
Figure 3.24 – Earth referential (E), Aircraft referential (A) and position vector P_E.

The following equation describes the position vector in Figure 3.24.

$$\mathbf{P}_E = [x \quad y \quad z]^T \tag{3.4.7}$$

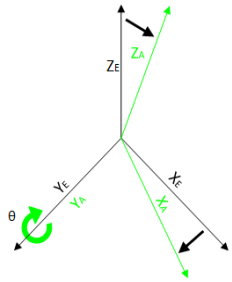
The angular position, or attitude from now on, is defined by the three rotations of the aircraft referential in relation to the earth referential, resulting in the roll (ϕ), pitch (θ) and yaw (ψ) already mentioned. The following three images show the rotations around each axis as well as the respective mathematical interpretation:

- Rotation around the X_E axis, giving the roll (ϕ) angle, defined by equation (3.4.8).



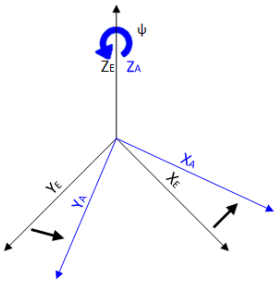
$$\mathbf{R}(\phi, x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.4.8}$$

- Rotation around the Y_E axis, giving the pitch (θ) angle, defined by equation (3.4.9).



$$\mathbf{R}(\theta, y) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.4.9}$$

- Rotation around the Z_E axis, giving the yaw (ψ) angle, defined by equation (3.4.10).



$$\mathbf{R}(\psi, z) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4.10}$$

To obtain the rotation matrix, we now must multiply the tree previous matrixes (3.4.8)· (3.4.9)·(3.4.10) resulting in $\mathbf{R}_\Theta$ represented in equation (3.4.11):

$$\mathbf{R}_\Theta = \mathbf{R}(\psi, z)\, \mathbf{R}(\theta, y)\, \mathbf{R}(\phi, x) \tag{3.4.11}$$

$$\mathbf{R}_\Theta = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \sin(\psi)\sin(\phi) - \cos(\psi)\sin(\theta)\cos(\phi) \\ -\sin(\psi)\cos(\theta) & \cos(\psi)\cos(\phi) - \sin(\psi)\sin(\theta)\sin(\phi) & \cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi) \\ \sin(\theta) & -\cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{3.4.12}$$

The linear velocity $\boldsymbol{\mathcal{V}}_A$ and the angular velocity $\boldsymbol{\Omega}_A$ are expressed by equations (3.4.13) and (3.4.14), respectively.

$$\boldsymbol{\mathcal{V}}_A = [u \quad v \quad w]^T \tag{3.4.13}$$

$$\boldsymbol{\Omega}_A = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T \tag{3.4.14}$$

Combining the linear and angular quantities, the representation of the position Γ and the velocity **ν** of the aircraft can be written as:

$$\boldsymbol{\Gamma} = [\mathbf{P}_E \quad \boldsymbol{\Theta}_E]^T = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T \tag{3.4.15}$$

$$\boldsymbol{\nu} = [\boldsymbol{\mathcal{V}}_A \quad \boldsymbol{\Omega}_A]^T = \left[u \quad v \quad w \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi}\right]^T \tag{3.4.16}$$

As equations (3.4.15) and (3.4.16) demonstrate, the open-loop system has twelve states – six states for positions and other six for angular and linear velocities. These states represent the six degrees of freedom of the open-loop system.

The next step is to obtain the dynamic model of the open-loop system, starting with the previous expressions. In order to do it, we must rely on the Newton-Euler equation. This method describes the combined translational and rotational dynamics of a rigid body (3.4.17), (Hahn, 2002).

$$\begin{bmatrix} \mathbf{F}_A \\ \mathbf{M}_A \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Omega}_A \times (\mathrm{m}\, \boldsymbol{\mathcal{V}}_A) \\ \boldsymbol{\Omega}_A \times (\mathbf{I}_{inert}\, \boldsymbol{\Omega}_A) \end{bmatrix} + \begin{bmatrix} \mathrm{m}\, \mathbf{I}_{ident} & \mathbf{N_3} \\ \mathbf{N_3} & \mathbf{I}_{inert} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\mathcal{V}}}_A \\ \dot{\boldsymbol{\Omega}}_A \end{bmatrix} \tag{3.4.17}$$

Where $\mathbf{I}_{inert}$ concerns the inertia on the three axis, m the mass, $\mathbf{N_3} \in \boldsymbol{\mathcal{R}}^{\mathbf{3\times 3}}$ is a null matrix and $\mathbf{I}_{ident}$ the three by three identity matrix. Solving the equation (3.4.17) step by step, simplifying each matrix separately:

- The matrix that results from solving the cross product of the vectors:

$$\begin{bmatrix} \boldsymbol{\Omega}_A \times (\mathrm{m}\, \boldsymbol{\mathcal{V}}_A) \\ \boldsymbol{\Omega}_A \times (\mathbf{I}_{inert}\, \boldsymbol{\Omega}_A) \end{bmatrix} = \begin{bmatrix} \mathrm{m}\, w\, \dot{\theta} - \mathrm{m}\, v\, \dot{\psi} \\ -\mathrm{m}\, w\, \dot{\phi} + \mathrm{m}\, u\, \dot{\psi} \\ \mathrm{m}\, v\, \dot{\phi} - \mathrm{m}\, u\, \dot{\theta} \\ I_{zz}\, \dot{\psi}\, \dot{\theta} - I_{yy}\, \dot{\theta}\, \dot{\psi} \\ -I_{zz}\, \dot{\psi}\, \dot{\phi} + I_{xx}\, \dot{\phi}\, \dot{\psi} \\ I_{yy}\, \dot{\theta}\, \dot{\phi} - I_{xx}\, \dot{\phi}\, \dot{\theta} \end{bmatrix} \tag{3.4.18}$$

- The matrix containing the mass and inertia of the aircraft on the XYZ axis:

$$\begin{bmatrix} \mathrm{m}\mathbf{I}_{ident} & \mathbf{N_3} \\ \mathbf{N_3} & \mathbf{I}_{inert} \end{bmatrix} = \begin{bmatrix} \mathrm{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathrm{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz} \end{bmatrix} \tag{3.4.19}$$

- The matrix containing the forces acting on the aircraft:

$$\begin{bmatrix} \mathbf{F}_A \\ \mathbf{M}_A \end{bmatrix} = [f_x \quad f_y \quad f_z \quad \mathrm{m}_x \quad \mathrm{m}_y \quad \mathrm{m}_Z]^T \tag{3.4.20}$$

The force acting on the open-loop system can be described by three components.

1. The first component corresponds to the action of the gravitational acceleration g [m· s^{-2}] on the aircraft. This only concerns the linear parameters, leaving aside any kind of angular factor. Equation (3.4.21) is the mathematical representation of the gravitational action.

$$\begin{bmatrix} \mathbf{F}_g^A \\ \mathbf{N}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_\Theta^{-1}\, \mathbf{F}_g^E \\ \mathbf{N}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_\Theta^T \begin{bmatrix} 0 \\ 0 \\ -\mathrm{m}\,\mathrm{g} \end{bmatrix} \\ \mathbf{N}_1 \end{bmatrix} = \begin{bmatrix} -\mathrm{m}\,\mathrm{g}\sin(\theta) \\ \mathrm{m}\,\mathrm{g}\cos(\theta)\sin(\phi) \\ -\mathrm{m}\,\mathrm{g}\cos(\theta)\,cos(\phi) \\ \mathbf{N}_1 \end{bmatrix} \tag{3.4.21}$$

Where $\mathbf{F}_g^A$ is the gravitational force regarding the aircraft referential, $\mathbf{F}_g^E$ is the gravitational force regarding the earth referential and $\mathbf{N}_1 \in \boldsymbol{\mathcal{R}}^{\mathbf{3\times 1}}$ is a null matrix. Another property that stands out is that the inverse rotation matrix is equal to its transposed because it is orthogonal normalized.

2. The second component correspond to the gyroscopic effect due to the thruster's rotation. This effect appears when the thrusters are rotation at a different speed and when the pitch and roll angle are not zero. Let this effect be **G** and the equation (3.4.22) represents its effect.

$$\mathbf{G} = \begin{bmatrix} \mathbf{N}_1 \\ -\sum_{i=1}^{8} \mathrm{J}_t \left(\boldsymbol{\Omega}_A \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) (-1)^{-i}\, \boldsymbol{\Omega}_i \end{bmatrix} = \begin{bmatrix} \mathbf{N}_1 \\ \mathrm{J}_t \begin{bmatrix} -\dot{\theta} \\ \dot{\phi} \\ 0 \end{bmatrix} \boldsymbol{\Omega} \end{bmatrix} \tag{3.4.22}$$

Where J_t is the inertial moment of rotation of the thrusters and $\boldsymbol{\Omega}$ [rad· s^{-1}] is the velocity matrix of the thrusters (3.4.23). As the equation (3.4.22) demonstrates, the gyroscopic effect is related only with angular parameters and not linear ones.

$$\boldsymbol{\Omega} = [\omega_1 \quad -\omega_2 \quad \omega_3 \quad -\omega_4 \quad -\omega_5 \quad \omega_6 \quad -\omega_7 \quad \omega_8]^T \tag{3.4.23}$$

3. The last component is related to the forces produced by the thrusters. Let $\mathbf{U}_A$ denote the vector where this effect is mathematically represented. In order to complete this effect, equations (3.4.1), (3.4.4), (3.4.5) and (3.4.6) are used to represent the angular velocities of the thrusters.

$$\mathbf{U}_A(\omega) = \begin{bmatrix} 0 \\ 0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{c}\left(\sum_{i=1}^{8} f_i\right) \\ \mathrm{c\,l}\,(f4 + f8 - f2 - f6) \\ \mathrm{c\,l}\,(f3 + f7 - f1 - f5) \\ \mathrm{d}\,(f1 + f3 + f6 + f8 - f2 - f4 - f5 - f7) \end{bmatrix} \tag{3.4.24}$$

Equation (3.4.24) already takes into account the distance of the thrusters to the center of mass l [m], the impulse coefficient c [N· s^2· rad^{-2}] and the drag coefficient d [N· m· s^2· rad^{-2}] generated by the rotation of the thrusters. Summing these three components we obtain:

$$\begin{bmatrix} \mathbf{F}_A \\ \mathbf{M}_A \end{bmatrix} = \begin{bmatrix} -\mathrm{m\,g}\,sin(\theta) \\ \mathrm{m\,g}\cos(\theta)\,\sin(\phi) \\ U_1 - \mathrm{m\,g}\,cos(\phi)\,cos(\theta) \\ U_2 - \mathrm{J}_t\,\omega\,\dot{\theta} \\ U_3 + \mathrm{J}_t\,\omega\,\dot{\phi} \\ U_4 \end{bmatrix} \tag{3.4.25}$$

With all three components, it is possible to rewrite equation (3.4.17) and simplify in order to $\begin{bmatrix} \dot{\boldsymbol{\nu}}_A \\ \dot{\boldsymbol{\Omega}}_A \end{bmatrix}$, obtaining the open-loop system of equations (3.4.26).

$$\begin{cases} \dot{u} = \left(v\,\dot{\psi} - w\,\dot{\theta}\right) - \mathrm{g}\sin(\theta) \\ \dot{v} = \left(w\,\dot{\phi} - u\,\dot{\psi}\right) + \mathrm{g}\cos(\theta)\sin(\phi) \\ \dot{w} = \left(u\,\dot{\theta} - v\,\dot{\phi}\right) - \mathrm{g}\cos(\theta)\cos(\phi) + \dfrac{U_1}{m} \\ \ddot{\phi} = \dfrac{1}{I_{xx}}\left(\left(I_{yy} - I_{zz}\right)\dot{\theta}\,\dot{\psi} - \mathrm{J}_t\,\dot{\theta}\,\omega + U_2\right) \\ \ddot{\theta} = \dfrac{1}{I_{yy}}\left(\left(I_{zz} - I_{xx}\right)\dot{\phi}\,\dot{\psi} - \mathrm{J}_t\,\dot{\phi}\,\omega + U_3\right) \\ \ddot{\psi} = \dfrac{1}{I_{zz}}\left(\left(I_{xx} - I_{yy}\right)\dot{\phi}\,\dot{\theta} + U_4\right) \end{cases} \tag{3.4.26}$$

Where ω corresponds to equation (3.4.27).

$$\omega = \omega_1 - \omega_2 + \omega_3 - \omega_4 - \omega_5 + \omega_6 - \omega_7 + \omega_8 \tag{3.4.27}$$

Equation (3.4.26) represents the open-loop system's dynamic in relation to the aircraft center of mass, in other words, to its referential. Now it is necessary to rewrite the linear parameters of the equation to the earth referential, as well as the angular parameters to the aircraft referential. The choice to represent the linear parameters in relation to the earth, and the angular parameters in relation to the aircraft, is just a way to simplify the identification of the control parameters. In order to aggregate the angular equations from one referential to the linear equations of the other, it was created a third referential K.

Equation (3.4.28) represents the velocity in the referential K:

$$\dot{\mathrm{Q}}_K = [\dot{\boldsymbol{P}}_E \quad \boldsymbol{\Omega}_A]^T = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix} \tag{3.4.28}$$

Rewriting equation (3.4.17) changing the aircraft's dynamic to the K referential, culminates in equation (2.3.6).

$$\begin{bmatrix} \mathbf{F}_E \\ \mathbf{M}_A \end{bmatrix} = \begin{bmatrix} \mathbf{N}_3 \\ \boldsymbol{\Omega}_A(\mathbf{I}_{inert}\,\boldsymbol{\Omega}_A) \end{bmatrix} + \begin{bmatrix} \mathrm{m}\,\mathbf{I}_{ident} & \mathbf{N_3} \\ \mathbf{N_3} & \mathbf{I}_{inert} \end{bmatrix} \begin{bmatrix} \dot{\mathrm{Q}}_K \end{bmatrix} \tag{3.4.29}$$

Now rewriting equations (3.4.18), (3.4.19), (3.4.20), (3.4.21) and (3.4.22) in order to K:

- Writing the cross product of the vectors results in:

$$\begin{bmatrix} \mathbf{N}_3 \\ \boldsymbol{\Omega}_A \times (\mathbf{I}_{inert}\ \boldsymbol{\Omega}_A) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_{zz}\,\dot{\psi}\,\dot{\theta} - I_{yy}\,\dot{\theta}\,\dot{\psi} \\ -I_{zz}\,\dot{\psi}\,\dot{\phi} + I_{xx}\,\dot{\phi}\,\dot{\psi} \\ I_{yy}\,\dot{\theta}\,\dot{\phi} - I_{xx}\,\dot{\phi}\,\dot{\theta} \end{bmatrix} \tag{3.4.30}$$

- The matrix containing the mass and inertia of the aircraft on the XYZ axis:

$$\begin{bmatrix} \mathrm{m}\,\mathbf{I}_{ident} & \mathbf{N_3} \\ \mathbf{N_3} & \mathbf{I}_{inert} \end{bmatrix} = \begin{bmatrix} \mathrm{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathrm{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz} \end{bmatrix} \tag{3.4.31}$$

This matrix is not modified, since the multirotor physical properties do not change.

- The matrix which concerns the gravity effect:

$$\begin{bmatrix} \mathbf{F}_g^E \\ \mathbf{N}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\mathrm{m}\,\mathrm{g} \\ \mathbf{N}_1 \end{bmatrix} \tag{3.4.32}$$

- The matrix containing the gyroscopic effect:

$$\mathbf{G} = \begin{bmatrix} \mathbf{N}_1 \\ \mathrm{J}_t \begin{bmatrix} -\dot{\theta} \\ \dot{\phi} \\ 0 \end{bmatrix} \boldsymbol{\Omega} \end{bmatrix} \tag{3.4.33}$$

This matrix, just like matrix (3.4.31), does not suffer any alteration. This happens because the thruster's rotation also influences the aircraft in the referential K.

The force produced by the thrusters will change where it concerns the U_1 in the new referential. This happens because of the relation between the earth and the aircraft's referentials. The new equation regarding the forces produced by the thrusters is:

$$\mathbf{U}_K(\omega) = \begin{bmatrix} \mathbf{R}_\Theta & \mathbf{N}_3 \\ \mathbf{N}_3 & \mathbf{I}_{ident} \end{bmatrix} \mathbf{U}_A(\omega) = \\ = \begin{bmatrix} U_1(sin(\phi)\, sin(\psi) - cos(\phi)\, cos(\psi)\, sin(\theta)) \\ U_1(cos(\psi)\, sin(\phi) + cos(\phi)\, sin(\psi)\, sin(\theta)) \\ \cos(\theta)\ \cos(\phi)\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.4.34}$$

Now adding the three components (3.4.32), (3.4.44), (3.4.34) and solving equation (3.4.29) in order to $\left[\dot{\Omega}_K\right]$, the result is the open-loop system of equations (3.6.1) that describes the aircraft's dynamic in relation to referential K.

$$\begin{cases} \ddot{x} = (sin(\phi)\, sin(\psi) + cos(\phi)\, cos(\psi)\, sin(\theta))\dfrac{U_1}{\mathrm{m}} \\ \ddot{y} = (cos(\psi)\, sin(\phi) - \ cos(\phi)\, sin(\psi)\, sin(\theta))\dfrac{U_1}{\mathrm{m}} \\ \ddot{z} = -\mathrm{g} + (cos(\phi)\, cos(\theta))\dfrac{U_1}{\mathrm{m}} \\ \ddot{\phi} = \dfrac{1}{I_{xx}}\Big(\big(I_{yy} - I_{zz}\big)\,\dot{\theta}\,\dot{\psi} - \mathrm{J}_t\,\dot{\theta}\,\omega + U_2\Big) \\ \ddot{\theta} = \dfrac{1}{I_{yy}}\Big(\big(I_{zz} - I_{xx}\big)\,\dot{\phi}\,\dot{\psi} - \mathrm{J}_t\,\dot{\phi}\,\omega + U_3\Big) \\ \ddot{\psi} = \dfrac{1}{I_{zz}}\Big(\big(I_{xx} - I_{yy}\big)\,\dot{\phi}\,\dot{\theta} + U_4\Big) \end{cases} \tag{3.4.35}$$

The kinematic and dynamic open-loop system is complete. To implement the open-loop system of equations above, some values must be calculated. The next subsection will demonstrate the computation of such values.

3.4.3 Complete X8-VB Quadcopter Model

3.4.3.1 Quadcopter Inertia

The quadcopter inertia takes an important part towards having an accurate model. The moments of inertia around x, y and z axis are represented by I_{xx}, I_{yy}, I_{zz}, respectively, as already encountered in the previous chapter (3.4.2 Kinematic and Dynamic Model). One must assume that the rotation angles roll, pitch and yaw do not change the inertial moments for any specific axis. As previously mentioned, the axis where placed in the aircrafts body overlapping the inertial rotation axis; this way, the calculation of the inertial moments is simplified, since the symmetry between the x and y axis is present.

In order to simplify the computations, one must consider that all mass components are solid cylinders attached by zero mass and frictionless arms. The inertial moment of a rotating cylinder in an axis perpendicular to its body is expressed by equation (3.4.36).

$$I = \frac{m\,r^2}{4} + \frac{m\,h^2}{12} \tag{3.4.36}$$

Where m is the cylinder's mass, r its radius and h its height.

Starting with the X-axis, the inertial moments of the cylinders representing the motors on either side of the axis (motors 2, 4, 6, 8) are approximated by the following equation (3.4.37).

$$I_{xx_1} = (4\,m)\,l^2 \tag{3.4.37}$$

Where m is the mass of a single cylinder and l refers to the length of both arms. Now looking at the motors in line with the axis (motor 1, 3, 5, 7) and the central part of the body, both affecting the inertia as equation (3.4.38) demonstrates.

$$I_{xx_2} = 4\left(\frac{m\,r^2}{4} + \frac{m\,h^2}{12}\right) + \frac{m_{cp}\,r_{cp}^2}{4} + \frac{m_{cp}\,h_{cp}^2}{12} \tag{3.4.38}$$

The first term of the equation regards the motors' influence. The latter terms are related with the aircraft center piece, where m_{cp} is the center piece mass and r_{cp}, h_{cp} are the radius and the height, respectively. These measures already take into account all the electronic and avionics equipment, as well as the battery. The overall moment of inertia I_{xx} can now be written as the sim of the expressions (3.4.37) and (3.4.38), resulting in the following equation (3.4.39).

$$I_{xx} = 4\left(\frac{m\,r^2}{4} + \frac{m\,h^2}{12}\right) + \frac{m_{cp}\,r_{cp}^2}{4} + \frac{m_{cp}\,h_{cp}^2}{12} + (4\,m)l^2 \tag{3.4.39}$$

Due to the symmetry mentioned before, the inertial moment computation about Y-axis follows a similar procedure.

The moment of inertia concerning the Z-axis is attributed to all eight motors and the center piece. The latter can be described as a cylinder rotating about an axis through and parallel to its center. This phenomenon is given by the equation (3.4.40).

$$I_{zz_1} = \frac{m_{cp}\,r_{cp}^2}{2} \tag{3.4.40}$$

For the eight motors at the end of the arms, the moment of inertia can be written as the equation (3.4.41).

$$I_{zz_2} = (8\,m)\,l^2 \tag{3.4.41}$$

Combining equations (3.4.40) and (3.4.41), the total inertial moment about the Z-axis can be written as their sum, resulting in the following equation (3.4.42).

$$I_{zz} = \frac{m_{cp}\,r_{cp}^2}{2} + (8\,m)\,l^2 \tag{3.4.42}$$

The results obtained from the measurements can be observed in Table 3.9.

Table 3.9 - Inertial calculations results.

INERTIA	VALUE
$\boldsymbol{I_{xx}}$	0.006 (N m s²)
$\boldsymbol{I_{yy}}$	0.006 (N m s²)
$\boldsymbol{I_{zz}}$	0.0166 (N m s²)

3.4.3.2 Quadcopter Drag

The quadcopter drag is related to its resistance to air. Drag is a fluid dynamic concept commonly known as the resistance to the air. It generates a force acting against the movement of an object, in this case the quadcopter (Falkovich, 2011). In order to include the drag force in the model, the open-loop system of equations (3.4.35) must be changed to add the f_d, resulting in the following system of equations (3.4.43).

$$\begin{cases} \ddot{x} = (sin(\phi)\ sin(\psi) + cos(\phi)\ cos(\psi)\ sin(\theta))\dfrac{U_1 - f_d}{\mathrm{m}} \\ \ddot{y} = (cos(\psi)\ sin(\phi) - cos(\phi)\ sin(\psi)\ sin(\theta))\ \dfrac{U_1 - f_d}{\mathrm{m}} \\ \ddot{z} = -\mathrm{g} + (cos(\phi)\ cos(\theta))\dfrac{U_1 - f_d}{\mathrm{m}} \\ \ddot{\phi} = \dfrac{1}{I_{xx}}\Big(\big(I_{yy} - I_{zz}\big)\ \dot{\theta}\ \dot{\psi} - \mathrm{J}_t\ \dot{\theta}\ \omega + U_2\Big) \\ \ddot{\theta} = \dfrac{1}{I_{yy}}\Big((I_{zz} - I_{xx})\ \dot{\phi}\ \dot{\psi} - \mathrm{J}_t\ \dot{\phi}\ \omega + U_3\Big) \\ \ddot{\psi} = \dfrac{1}{I_{zz}}\Big(\big(I_{xx} - I_{yy}\big)\ \dot{\phi}\ \dot{\theta} + U_4\Big) \end{cases} \tag{3.4.43}$$

Where f_d is:

$$f_d = \frac{1}{2}\rho\ v^2\ \mathrm{C_d}\ \mathrm{A} \tag{3.4.44}$$

The equation (3.4.44) is dependent of the speed v, the cross section area A, the density of the air ϱ and the drag coefficient $\mathrm{C_d}$.

3.4.3.3 BLDC motor & Propeller

The final step in the model creation is the motor's and propeller's physical description. In order to obtain the model, experimental testes took place. The use of experimental data was needed because the motor's and propeller's manufacturer does not provide enough technical specifications. The motors used, as mentioned before, were brushless DC (BLDC) working at 14.8 V and controlled by the electronic speed controllers (ESC), shown *supra*, which are actuated with PWM signals.

A test bench was built for the experiments and the monuments were acquired with different dynamometers (Figure 3.25). The testing was divided into six levels of speed. This is done in order to keep the electronics' and motor's integrity. These six levels can be reorganized in three: low thrust, medium thrust and high thrust.

Figure 3.25 – Developed and implemented test bench.

The results of the experiments can be observed in Figure 3.26 and Figure 3.27. The graphics contain not only the experimental data segmented by levels of speed, as mentioned before, but also a polynomial regression approximation obtain from the data.

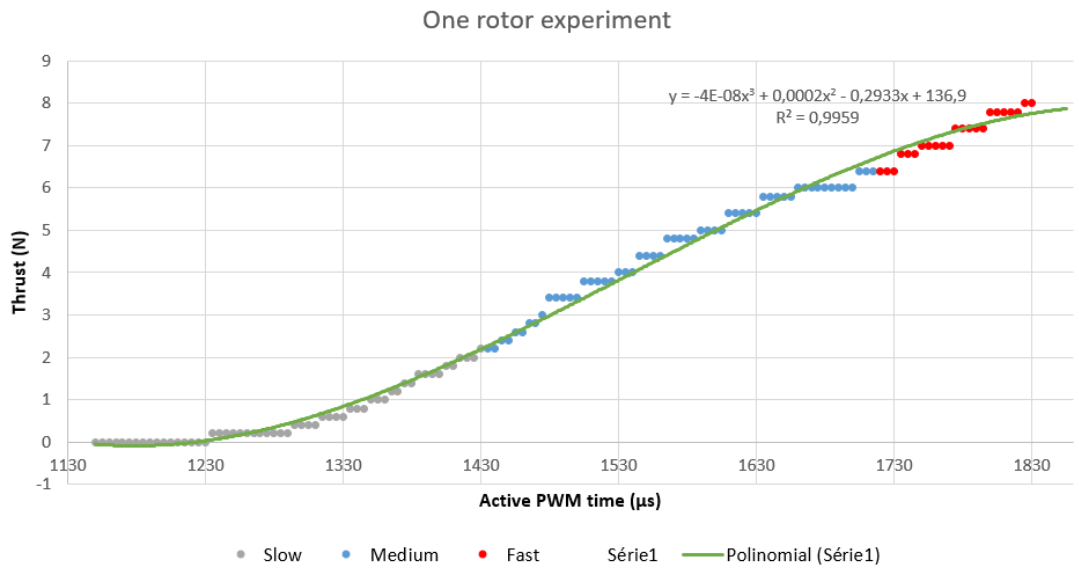


Figure 3.26 - Motor and propeller experimental result.

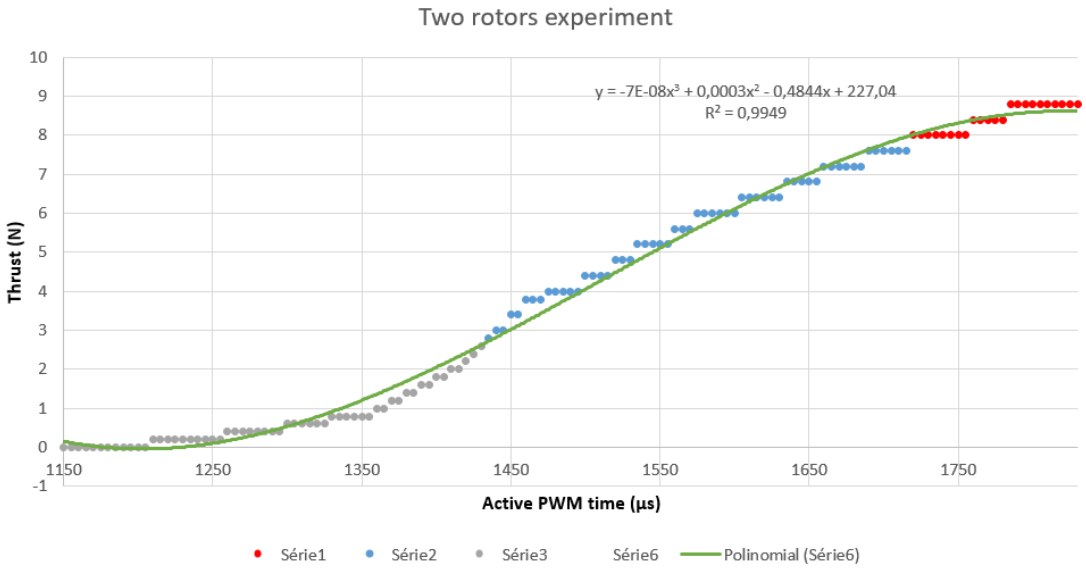


Figure 3.27 – Two motors and propellers experimental result.

From the results, one must conclude that there is no improvement regarding thrust with this architecture, since stability and actuators redundancy, in order to tolerate faults, were the main goals, this is no issue to the final objective.

Thrust was not the only data collected from the experiments. Current and speed were also collected, in order to fill the model with accurate data. The following system of equations (3.4.45) shows the current (i) and speed (ω) of each motor collected and implemented in the model.

$$\begin{cases} i = 2{\times}10^{-5}\, q^2 - 0.0418\, q \; + \; 22.501 \text{ (A)} \\ \omega = 432.29\, q^2 - 1779.3\, q + 1993.5 \text{ (rad s}^{-1}\text{)} \end{cases} \tag{3.4.45}$$

The input q in the system of equations (3.4.45) represents the active PWM time used to manipulate the ESC. In order to optimize execution time, only the experiments of one motor were used to create an ARX model. The ARX(na, nb, nk) model was a

$$f_n(s) = \frac{47.6205}{s^2 + 25.9384\, s + 47.6205} \tag{3.4.46}$$

ARX(2,2,1) and then converted to transfer function, resulting in equation (2.3.6) that represents the force in each arm.

The transfer function step response can be observed in Figure 3.28.

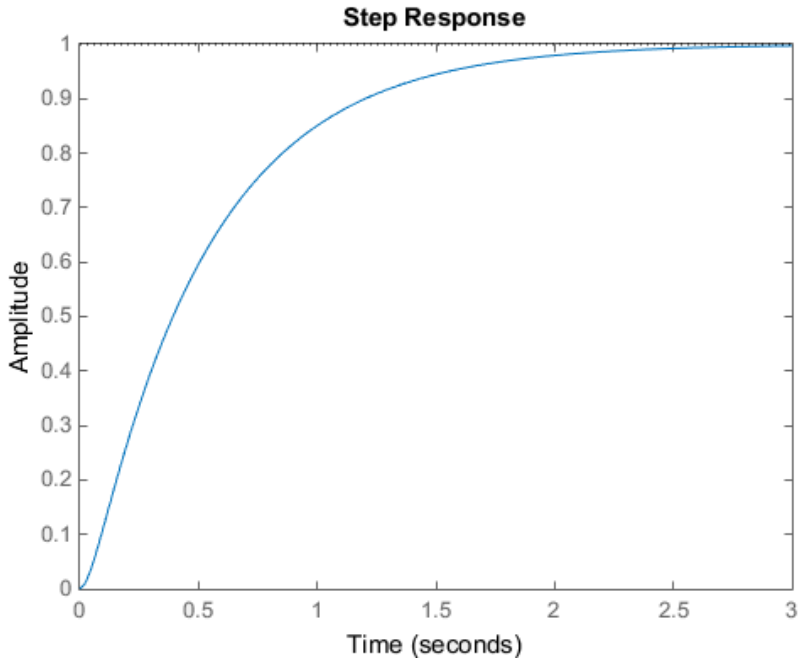


Figure 3.28 - Motor model step response.

3.4.3.4 Connection of the Model to the Simulator

The modeling phase reached the end with the results from the motors experiments. The equations and results demonstrated before were implemented in Matlab/Simulink (represented in the attachment B. Simulink Model) connected to a virtual reality world developed with didactic and research purposes (attachment C. Virtual World). Next, the control architectures and design approaches will be described.

3.5 Control Architectures and Design

This section contains the control architectures that were implemented in this research work. First, the architectures were implemented using the model described before. Subsequently, some of these architectures were implemented in the real aircraft, contemplating some differences due to sampling time and sensors response.

The architectures presented have as their goal the maintenance of the flight's stability. Therefore, the aircraft must have some operational specifications to its movements - the pitch and roll rotations are limited in the interval [-20; 20] degrees and their controllers, as well as the yaw controller, must have an actuation limited between [-50; 50] of the total PWM range.

3.5.1 High Level Control Architecture

The high level control architecture is represented in Figure 3.29. As mentioned before, each actuator has its own controller (ESC) that will be managed by an on-board supervisor that handles their reconfiguration, not only in case of fault/failure, but also according to the mission's goal. In case of communication losses, the on-boar supervisor will handle critical situations. For instance, if an actuator fails, the on-board supervisor will land immediately and safely.

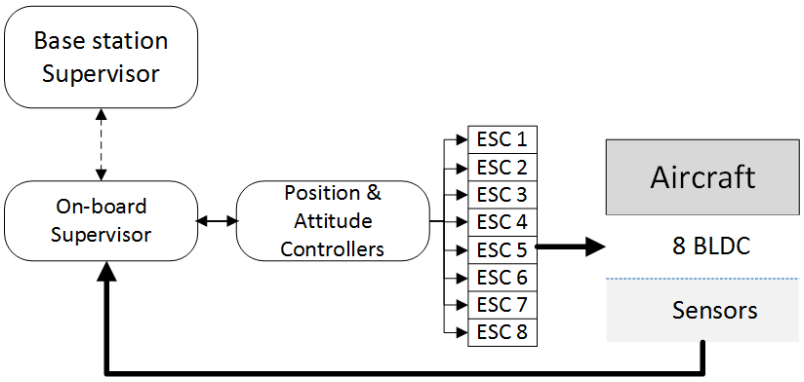


Figure 3.29 - High-level control architecture.

The base-station supervisor intent is to receive the real time flight data detecting faults/failures on actuators and/or sensors. It will also provide user information, as well as allow user input, in order to manipulate route or mission.

In case of a fault/failure, the base station is capable of reconfiguring the system, in order to fulfill the mission. For example, if a motor fails, the base station is capable of manipulating the controllers, in order to keep the aircrafts stability and flight.

3.5.2 Low Level Control Architectures

The low level control architecture holds up to the control itself. The architecture is shown in Figure 3.30 and represents the controllers actuating in the eight motors, according to equations (3.4.1), (3.4.2), (3.4.3) and (3.4.4). It is embedded in Figure 3.29 in the Position & Attitude Controllers block. The main controller is the altitude controller and its actuation is manipulated by the pitch, roll and yaw controllers before entering the ESCs. As already mentioned, the actuation from the rotation controllers is between [-50; 50], this way the motors never shut down in order to rotate, assuring the stability of the flight.

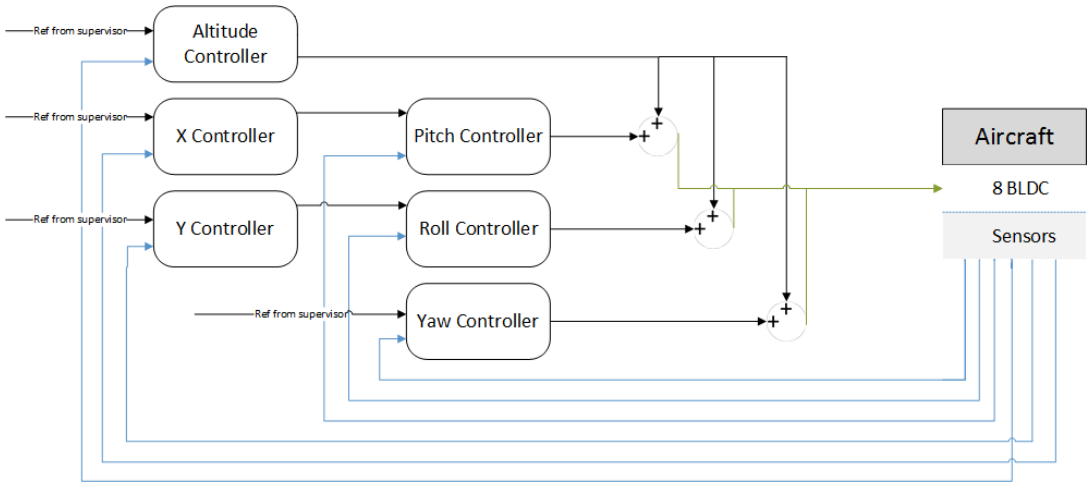


Figure 3.30 – Low-level control architecture.

The setpoint of the pitch and roll controllers comes from the position controllers and the yaw controller receives the setpoint from the supervisor (on-board supervisor), in order to turn the heading to the waypoint. The position controllers receive the setpoint from the same supervisor.

The actuations on the motors from the roll, pitch, yaw and altitude controllers follow the subsequent equations (3.5.1).

$$\begin{aligned} u_{m1} &= u_{altitude} - u_{pitch} + u_{yaw} \\ u_{m2} &= u_{altitude} - u_{roll} - u_{yaw} \\ u_{m3} &= u_{altitude} + u_{pitch} + u_{yaw} \\ u_{m4} &= u_{altitude} + u_{roll} - u_{yaw} \\ u_{m5} &= u_{altitude} - u_{pitch} - u_{yaw} \\ u_{m6} &= u_{altitude} - u_{roll} + u_{yaw} \\ u_{m7} &= u_{altitude} + u_{pitch} - u_{yaw} \\ u_{m8} &= u_{altitude} + u_{roll} + u_{yaw} \end{aligned} \tag{3.5.1}$$

3.5.3 Control Design Based on Simulations

The controllers' gains were first obtained from the simulations using Ziegler-Nichols and Åström's relay techniques and then by means of an optimization algorithm. The first techniques were applied directly to the Simulink model. On the other hand, the optimization algorithm was already developed on Matlab script and the model was adapted to work with it.

The following topic (3.5.3.1) describes the results from the Ziegler-Nichols and Åström's relay techniques to find the controllers' gains. Subsection 3.5.3.2 contains the optimization algorithm implementation and results.

3.5.3.1 Control tuning based on Ziegler-Nichols and Åström approaches

The control algorithm implemented was a PID controller, since these controllers assure the stability of closed-loop systems. In the case of a quadcopter, the PID controller must be switched according to some parameters (for instance altitude) because the open-loop system has a non-linear behavior.

The PID controllers follows the same architecture represented in Figure 3.30. The PID architecture is shown in equation (2.3.1) and it is represented in Figure 3.31, where an anti-windup is added in order to counteract the integral error accumulation.

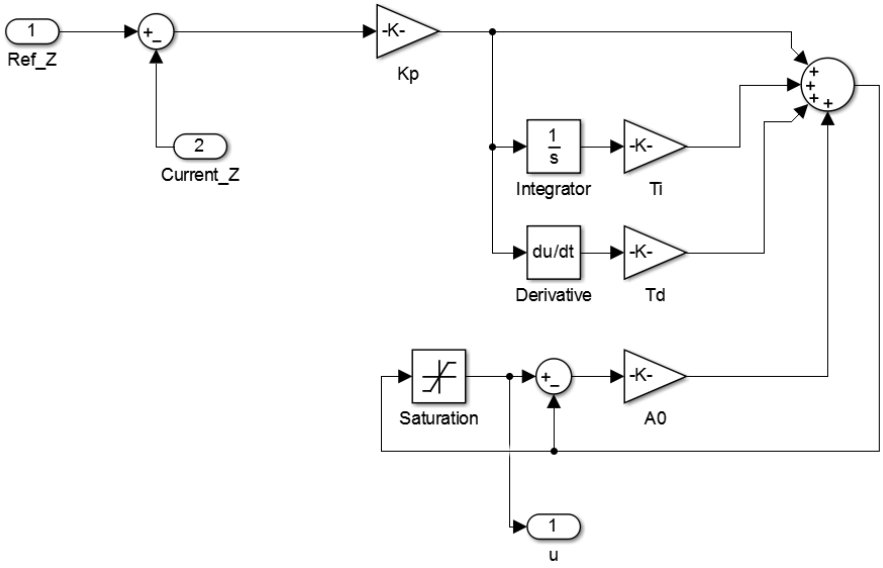


Figure 3.31 - PID with anti-windup architecture.

The technique used to tune the gains of some controllers was the relay method (Karl Johan Åström & Hägglund, 2006). It follows the equation (3.5.2).

$$K_u = \frac{4\,\Delta}{\pi\sqrt{a^2 - \varepsilon^2}} \tag{3.5.2}$$

Where Δ corresponds to the actuation's amplitude used in the relay, a is the amplitude of the resulting oscillation and ε represents the hysteresis of the actuation. The results obtained from the relay method are then applied to the Ziegler-Nichols rules (Ziegler & Nichols, 1942) where the controller's gains are obtained. Table 3.10 was the one used to obtain the controller gains.

Table 3.10 - Ziegler-Nichols Rules

Type	K_p	T_i	T_d
P	$0.5\ K_u$		
PI	$0.45\ K_u$	$\frac{T_u}{1.2}$	
PID	$0.6\ K_u$	$\frac{T_u}{2}$	$\frac{T_u}{8}$

The other technique used to tune the controllers was the ultimate sensitivity. This method consists in using a proportional gain to make the closed-loop system stay in a permanent oscillation (Ziegler & Nichols, 1942). The proportional gain is considered to be K_u and the period of oscillation is the T_u used to compute the gains in accordance with the Ziegler-Nichols table presented *supra* (Table 3.10).

A. Altitude controller

The first controller tuned and implemented was the altitude controller, as it is the main controller setting the actuation reference. The Switching PID gains were adjusted every 20 meters. This was made in order to eliminate overshoot and minimize the raising time. The results from the relay application can be observed in Figure 3.32 where the data tips presented in the graphics indicate the values used to calculate the controllers' gains.

For the final implementation in the real aircraft, only the controller from the first 20 meters was implemented, since it was considered an altitude high enough to assure the safety of the prototype.

The switching of the controllers was not considered a case of study, since the simulation presented good performance in switching. For the real prototype, the switching needed to be smooth to prevent the control to become unstable. Just as the switching of the controllers, the dwell-time (minimal time before switch) was not revised, since the prototype was not implemented with it and this approaches was abandoned for now.

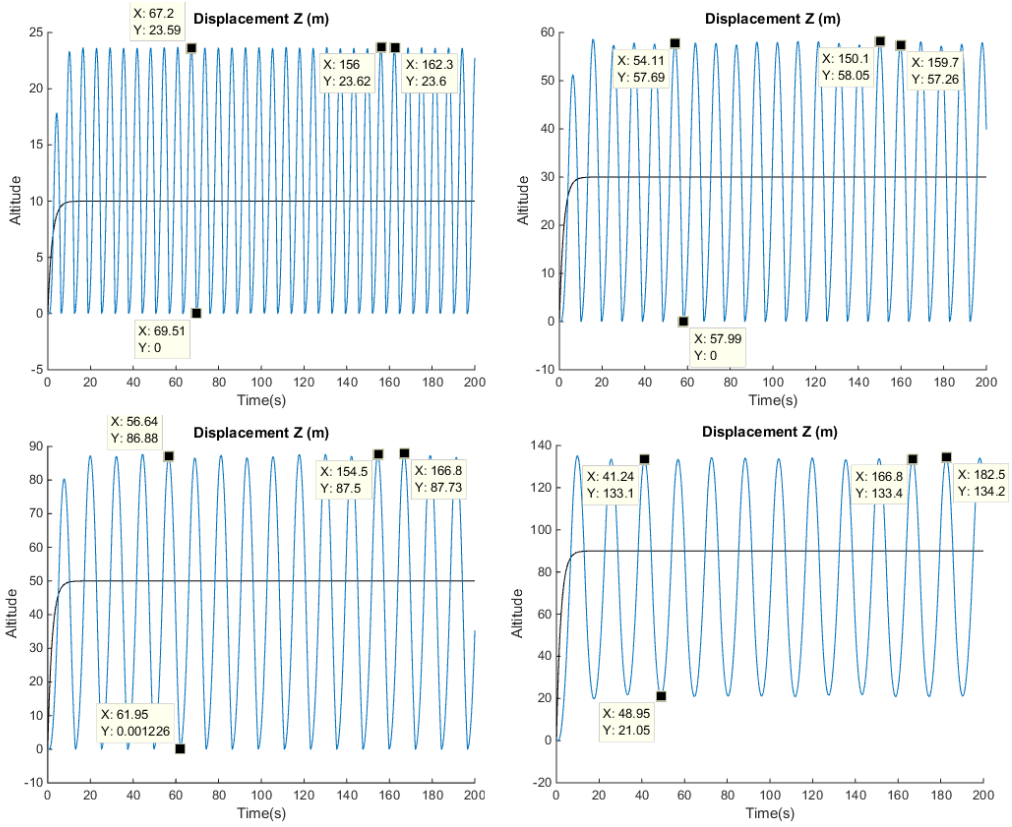


Figure 3.32 - Relay controller application to the altitude. The black line represents the setpoint and the blue one the closed-loop system's response.

The gains resulting from the relay method can be observed in Table 3.11, with some slight differences from the exact calculations. The difference lays on the fact that Ziegler-Nichols rules are generic and the gains obtained from there need to be adjusted according to the closed-loop system behaviour.

Table 3.11 - Altitude PID gains.

Altitude (m)	K_p	T_i	T_d
0-20	33	10	6
20-40	10	5	1.2
40-60	6	6	1.53
80-100	5	8	2

The ground effect is a well-known problem and it is somehow contemplated in the controller that handles the lift off and landing. The ground effect represents the interruption of the downwash of the air below the propellers, increasing the lift of the aircraft (Dole & Lewis, 2000). To lift off the motors should be at the max speed for some time and to land there are some possible maneuvers. One possible way to cover this effect is to maintain a certain horizontal speed until the touchdown; another is to just shut down the motors when the proximity to the ground is enough to prevent structural damage.

B. Pitch / Roll controllers

The pitch and roll rotations have the same PID architecture. However, in order to obtain the gains of the controller, it was not possible to use the relay method as it was for the altitude controller described *supra*. The relay led the closed-loop system to instability, consequently, the ultimate sensitivity method was used in these two cases. The results from the permanent oscillation can be observed in Figure 3.33.

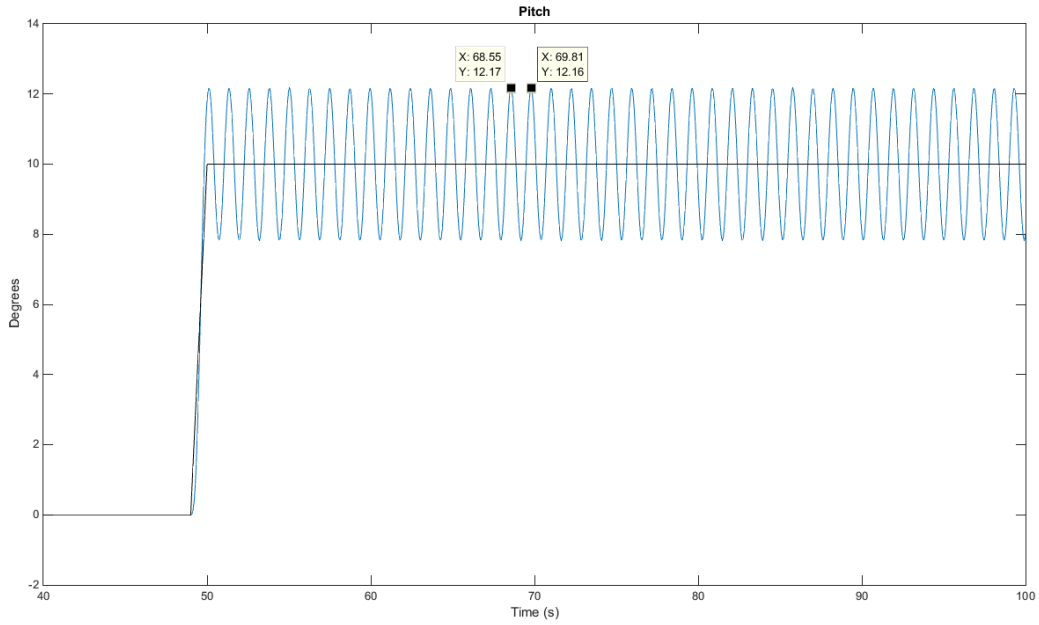


Figure 3.33 - Ultimate sensitivity method applied to pitch rotation.

Using the data tips, one can calculate the period and obtain the gains of the controller for pitch and roll rotation shown in Table 3.12. As mentioned before, the gains that were implemented suffered slight adjustments, in order to keep some specifications. The controller obtained from the pitch rotation was also applied to the roll rotation, since the symmetry around the axis exists and allows the closed-loop system to behave in the same way.

Table 3.12 - Roll/Pitch PID gains.

Rotation	K_p	T_i	T_d
Roll	0.1	5	7
Pitch	0.1	5	7

These controllers have an actuation limitation to maintain a steady flight. As referenced before, the actuation can only vary between [-50; 50] and this value is added or subtracted to the actuation of the altitude controller.

C. *Yaw controller*

The yaw controller has the same problem as the pitch and roll controllers, which is the relay method not working properly, making the closed-loop system unstable. For this reason, the ultimate sensitivity was also used to obtain the gains for the yaw controller. The rotation can vary from [0; 360] in a positive or negative way.

The result from the application of the ultimate sensitivity can be observed in Figure 3.34.

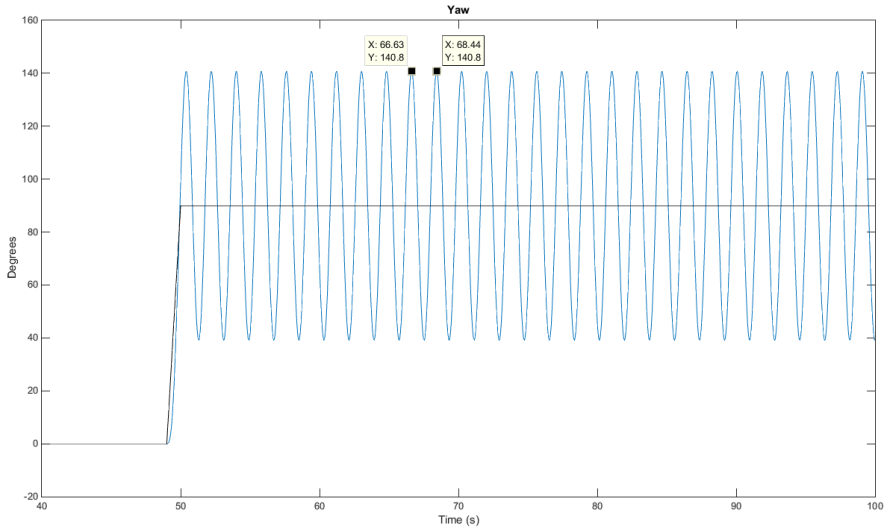


Figure 3.34 - Ultimate sensitivity method applied to yaw rotation.

As usual, the data tips allow the calculation of the period, permitting the controller's gain calculation to take place. The results from the ultimate sensitivity can be observed in Table 3.13.

Table 3.13 - Yaw PID gains.

Rotation	K_p	T_i	T_d
Yaw	0.78	4	2.5

This controller follows the same specification as the above, hence, its actuation can only vary between [-50; 50], in order to maintain the stability and reliability of the controller and the flight.

At this point, the gains to the attitude controllers are calculated. Subsequently, the position and trajectory controllers were obtained using the same PID architecture.

D. Position and Trajectory Controller

The position and trajectory controllers have the same PID architecture described before. The hardest part of tuning these controllers relates to the fact that they are based on cascade control scheme (Figure 3.35). The X and Y positions' controllers send their actuation as the setpoint to the roll and pitch controllers. The tuning process of such control scheme follows specific steps:

1- Regulate the inner loop controller according to its process variable;
2- Set the inner loop controller to automatic with internal setpoint;
3- Adjust the outer loop according to its process variable;
4- Switch the inner loop controller to the external setpoint;
5- Shift the outer loop to the automatic mode.

If the steps presented before are not followed properly, the closed-loop system may suffer from undesired switching transients (Karl Johan Åström & Hägglund, 1995). The first two steps were followed in the previous section Pitch / Roll controllers. The subsequent steps shall now take place.

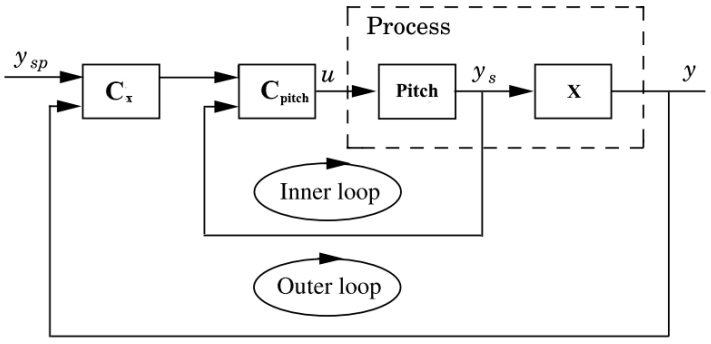


Figure 3.35 - Block diagram example of the cascade control system. Adapted from K. Åström & T. Hägglund, 1995.

The trajectories were considered to be linear and with no obstacles. There are two types of possible trajectories maneuvers. The first is the simple way were the yaw rotation is not manipulated, the roll and pitch rotation move the aircraft to the intended point in space without ever turning the heading, good for some applications such as shooting sideways a moving object.

The other way to manipulate the trajectory is to operate the yaw, in order to move the heading of the aircraft to the reference point, using only the pitch rotation to move forward/backward and the roll rotation to adjust the course in case of disturbances, good for some applications such as shooting a moving object from behind. This method requires a supervisor to manipulate the setpoints of the controllers.

Since the two movements are symmetrical (pitch to manipulate X and roll to manipulate Y) the controller adjustment was adjusted using only the X variable, just as it was done for the rotations pitch and roll. The method used to obtain the controller's gains was the ultimate sensitivity, also mentioned before. The results from this technique can be observed in Figure 3.36.

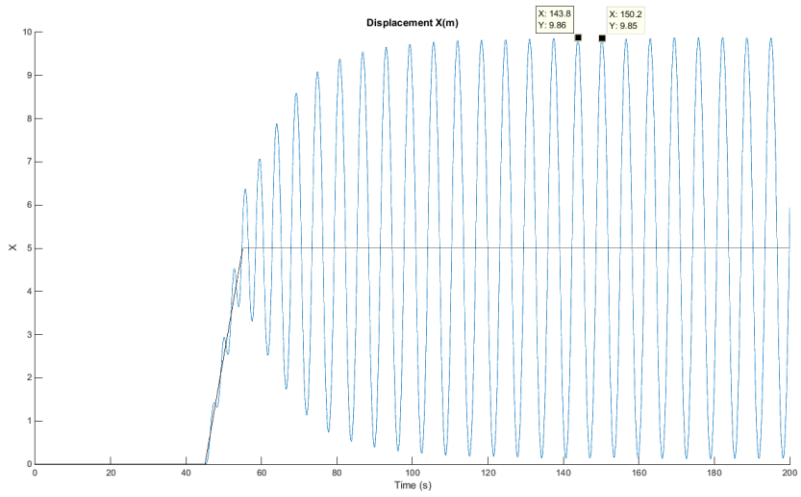


Figure 3.36 - Ultimate sensitivity applied to the X variable.

The results of the gains obtained from the two methods can be observed in Table 3.14.

Table 3.14 - X/Y PID gains.

Variable	K_p	T_i	T_d
X	0.3	24	5
Y	0.3	24	5

The control gains are the same regardless of what is the type of trajectory maneuvering desired. In order to manipulate trajectory without manipulating yaw rotation, the setpoint of the controllers are only the target coordinates. To operate trajectory with yaw rotation, some operations to the setpoints take place. For instance, if the reference point is (5, 5) meters from the current position it means that yaw rotation should be 45º, as Figure 3.37 demonstrated.

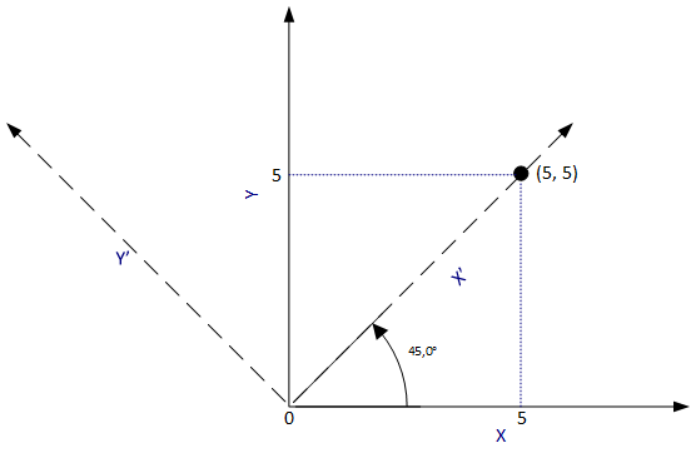


Figure 3.37 - Trajectory with yaw rotation.

To manipulate the axis according to the waypoint, the yaw setpoint (sp_{yaw}) is calculated by the following equation.

$$sp_{yaw} = -\tan^{-1}\frac{Y_{ref} - Y_{init}}{X_{ref} - X_{init}} \tag{3.5.3}$$

Where Y_{ref} and X_{ref} are the intended waypoint and Y_{init} and X_{init} are the initial position. After the yaw setpoint is calculated, the X and Y controllers' setpoints need to be regulated as well. The way to do it is to use the sp_{yaw} and calculate the new setpoint and rectify the current value also. The formulas that calculate these parameters are the subsequent.

$$sp_x = \left(X_{ref} - X_{init}\right)\cos\left(sp_{yaw}\right) - \left(Y_{ref} - Y_{init}\right)\sin\left(sp_{yaw}\right) \tag{3.5.4}$$

$$sp_y = \left(X_{ref} - X_{init}\right)\sin\left(sp_{yaw}\right) + \left(Y_{ref} - Y_{init}\right)\cos\left(sp_{yaw}\right) \tag{3.5.5}$$

The current value is adjusted with the same equations (3.5.4) and (3.5.5), but instead of X_{ref} and Y_{ref} the current position is included.

The Ziegler-Nichols and Åström's relay methods of designing the controllers reaches the end with the position and trajectory controller. The simulations and results can be observed in Chapter 4, specifically in 4.1 Simulations. The optimization algorithm used to find the controllers' gains will now be presented.

3.5.3.2 Control tuning based on PSO

In this research, the classical Particle Swarm Optimization (PSO) explained in the previous chapter was used to obtain the controllers' gains. For a control structure selected previously, the optimization problem is to find the control parameters that minimize a cost function. For this work, the optimization was done on-line with the simulation model. To keep in mind that the results from the optimization process are always sub-optimal solutions since, in order to obtain optimal results, a near infinite loop was required.

The model suffered alterations because a discretization was in order to apply the optimization algorithm. The continuous model progress is made by an integration method provided by the ode45 function in Matlab. The integration step was stablished to a tenth of the sampling time and the output data from the integration process is made available in each sampling time instant, the sampling time used for this method was $T_s = 100ms$.

The optimization algorithm has two parts: the first half of the simulation is where the algorithm takes place and the controllers are tested; the second half is where a simulation with the controller that minimized the cost function is tested. The optimizations were made considering five particles with ten iterations each. Its duration was 2000s leaving 20s for each controller to be active.

The cost function used for both controllers contains the control mean-squared error (mse) and the control variance (var). The weigh distribution was represented in equation (3.5.6).

$$J_c(k) = \frac{1}{2}\, mse\big(e(k-n+1:\ k)\big) + \frac{1}{2}\, var(u(k-n+1:\ k)) \tag{3.5.6}$$

Where $(k-n+1:k)$ represents the set of samples of the active control in a window. The cost function is calculated at the end of the active time of each controller.

A. PID controller

The PID algorithm implemented was discretized in order to be applied to the optimization process. The discrete PID control algorithm was based on the book *Computer Control System - Theory and Design* of K. J. Åström & Wittænmark, 1997. The controllers take into account the sampling time and contemplate the anti-windup filter. The algorithm is presented by the following set of equations.

$$e(k) = r(k) - y(k) \tag{3.5.7}$$

$$bi = \frac{K_p\, T_s}{T_i} \tag{3.5.8}$$

$$ad = \frac{2\, T_d - N\, T_s}{2\, T_d + N\, T_s} \tag{3.5.9}$$

$$bd = \frac{2\, K_p\, N\, T_d}{2\, T_d + N\, T_s} \tag{3.5.10}$$

$$ao = \frac{T_s}{T_t} \tag{3.5.11}$$

$$P(k) = K_p\, e(k) \tag{3.5.12}$$

$$D(k) = ad\, D(k-1) - bd\, \big(y(k) - y(k-1)\big) \tag{3.5.13}$$

$$v(k) = P(k) + I(k-1) + D(k) \tag{3.5.14}$$

$$u(k) = sat(v(k), ulow, uhigh) \tag{3.5.15}$$

$$I(k) = I(k-1) + bi\, e(k) + ao(u(k) - v(k)) \tag{3.5.16}$$

Where $e(k)$ is the error, $r(k)$ the setpoint, $y(k)$ the sensor data, bi the integral gain, ad and bd derivatice gains, ao the anti-windup gain where $T_t = T_i$, $P(k), D(k),$ and $I(k)$ the PID values, $v(k)$ the temporary control action and $u(k)$ the applied control action.

The optimization was made for all six variables and an example for the altitude can be observed in Figure 3.38. Altitude represents the setpoint and closed-loop system's response, U_1 is the result of the motors force, G1, G2 and G3 represent K_p, T_i and T_d, respectively, Ha is the Harris index, Co is the cost function result for each controller, Act is the active controller and Bct is the best controller at each instance. From 1000s, the controller that minimized the cost function (the best controller) is activated and simulated until the end of the experience; in this case, it was the 38th controller with a cost function result of 0.00020131.

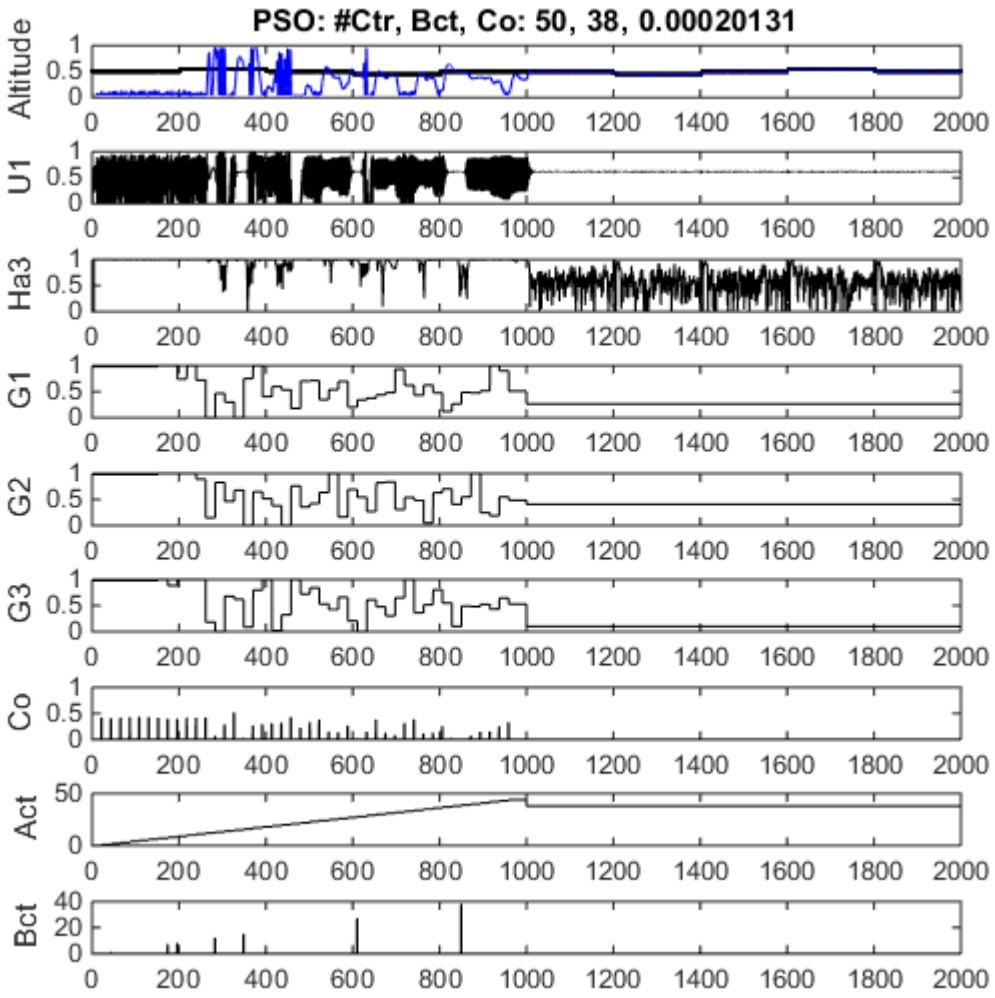


Figure 3.38 - Example of the optimization process applied to altitude.

The values are normalized between [0; 1] which represent [0; 20] (m) and even the controllers' actuations and gains are normalized, being multiplied by a gain before entering the model. The gains resulting from the optimization can be observed in Table 3.15.

Table 3.15 - PID controllers' gains obtained from PSO algorithm.

variable	K_p	T_i	T_d
Z	2.72	4.22	1.03
Y	4.58	36.00	10.01
X	4.58	36.00	10.01
Roll	1.10	7.64	3.51
Pitch	1.10	7.64	3.51
Yaw	5.55	6.23	2.40

B. SMC controller

The algorithm implemented in this work is a PID type sliding surface (Brito Palma et al., 2015). The discrete control architecture follows the equations presented next.

$$e(k) = r(k) - y(k) \tag{3.5.17}$$

$$d_e(k) = g_e \frac{\big(e(k) - e(k-1)\big)}{T_s} + p_c d_e(k-1) \tag{3.5.18}$$

$$g_{aw} = \frac{T_s \lambda}{c} \tag{3.5.19}$$

$$i_e(k) = i_e(k-1) + T_s e(k) + g_{aw}\left(u(k-1) - v(k-1)\right) \tag{3.5.20}$$

$$\sigma_c(k) = c\, e(k) + \lambda\, i_e(k) + d_e(k) \tag{3.5.21}$$

$$v(k) = \rho_c \frac{\sigma_c(k)}{|\sigma_c(k)| + \epsilon_c} \tag{3.5.22}$$

$$u(k) = sat(v(k), [0; 1]) \tag{3.5.23}$$

The controller error is given by equation (3.5.17), the derivative term defined by equation (3.5.18), the integral component expressed by (3.5.20), the integral component with anti-windup term given by equation (3.5.21), the temporary control action defined by equation (3.5.22) and the smooth saturated control action described by equation (3.5.23). The parameters present in the controller are the error gain g_e, the high-pass filter pole p_c, the sampling time T_s, the anti-windup gain g_{aw}, the speed parameter c, the integral gain λ and the stability parameter ρ_c.

The same process was used to tune the SMC gains. Optimization algorithm was implemented to altitude as before. As the PID controller, all the values are normalized between [-1, 1]. The resulting gains from the optimization can be observed in Table 3.16.

Table 3.16 - SMC controllers' gains obtained from PSO algorithm.

variable	$\boldsymbol{c}$	$\boldsymbol{\lambda}$	$\boldsymbol{g_e}$	$\boldsymbol{\rho_c}$	$\boldsymbol{\epsilon_c}$	$\boldsymbol{p_c}$
Z	7.11	7.73	16.40	5.39	18.71	0.25
Y	6.84	0.025	19.83	10.86	9.31	0.47
X	6.84	0.025	19.83	10.86	9.31	0.47
Roll	4.36	2.01	9.76	9.38	19.93	0.10
Pitch	4.36	2.01	9.76	9.38	19.93	0.10
Yaw	12.09	4.16	15.86	18.09	19.61	0.52

Simulations and analysis of the controllers will be presented in the 4.1. Simulations. Next, the fault / failure study and purposed re-configurations will take place.

3.6 Fault Tolerant Control

The fault tolerance is enhanced by the proposed architecture, since the redundancy of actuators (motors) and sensors exists. The controllers have the same architecture presented before but, in case of actuators fault, some adjustments take place. The sensors faults were not studied due to the lack of time and the complexity of designing an observer with that purpose.

The question in hands relates to the fact of what happens if a motor stops working. The results can be seen by the examples *infra*.

If a motor fails in a quadcopter with only four motors, the balance between the forces will not be tolerated. The following system of equations demonstrate it.

$$\begin{cases} U_1 = f_1 + f_2 + f_3 + f_4 \\ U_2 = f_4 - f_2 \\ U_3 = f_3 - f_1 \\ U_4 = f_1 + f_3 - f_2 - f_4 \end{cases} \tag{3.6.1}$$

The sums of forces represented in equation (2.3.6) must be zero, except U_1 which is the altitude related force. If, for instance, motor 1 (f_1) fails, the result is the following system of equations.

$$\begin{cases} U_1 = f_2 + f_3 + f_4 \\ U_2 = f_4 - f_2 \\ U_3 = f_3 \\ U_4 = f_3 - f_2 - f_4 \end{cases} \tag{3.6.2}$$

Considering that all forces are constants (aircraft is in hoovering), the balance between forces is not the same. The result can be observed in Figure 3.39 where the pitch rotation is clearly oscillatory and there is no control action or reconfiguration to recover from it, that is, balance U_3.

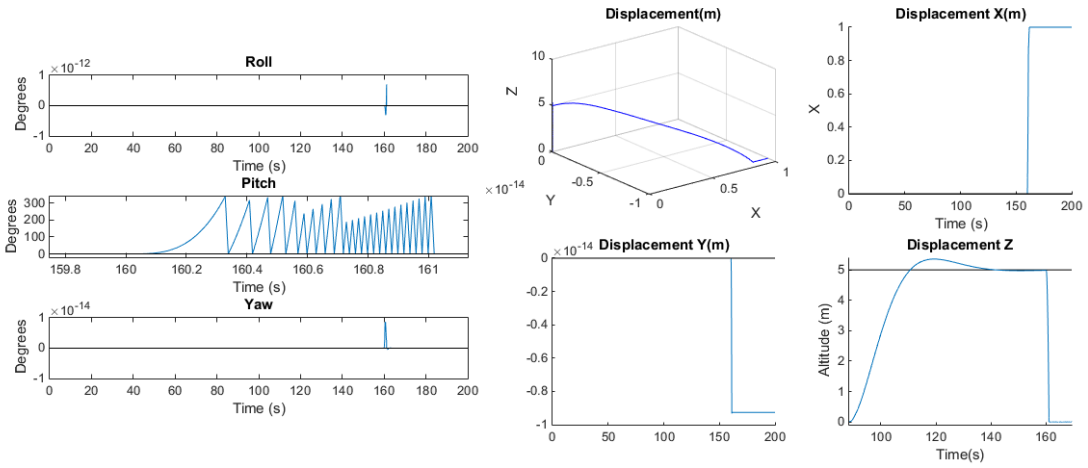


Figure 3.39 - Four motors quadcopter fault experiment.

The purposed architecture, on the other hand, can tolerate up to four faulty motors, depending on which ones. The tolerance resorts to some system reconfiguration as will be demonstrated next.

Using the same example as before, adding the additional motors to the system of equations (2.3.6) the following system of equations used in the model explained before is obtained.

$$\begin{cases} U_1 = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \\ U_2 = f_4 + f_8 - f_2 - f_6 \\ U_3 = f_3 + f_7 - f_1 - f_5 \\ U_4 = f_1 + f_3 + f_6 + f_8 - f_2 - f_4 - f_5 - f_7 \end{cases} \tag{3.6.3}$$

Taking the same fault from the previous example, in case of motor 1 (f_1) failure, the result would be represented in the subsequent system of equation.

$$\begin{cases} U_1 = f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \\ U_2 = f_4 + f_8 - f_2 - f_6 \\ U_3 = f_3 + f_7 - f_5 \\ U_4 = f_3 + f_6 + f_8 - f_2 - f_4 - f_5 - f_7 \end{cases} \tag{3.6.4}$$

Looking at the system as an optimization problem, as the aircraft is in flight (hoovering), the sum of forces should be zero, except U_1. With this knowledge, by fixing the forces produced by the motors from the arm that did not suffer the fault (f_2, f_4 , f_6 and f_8) and solving the system, the results demonstrate that the motor from the opposite site should be deactivated, in this case motor 7 (f_7). The result of this reconfiguration can be seen in Figure 3.40.

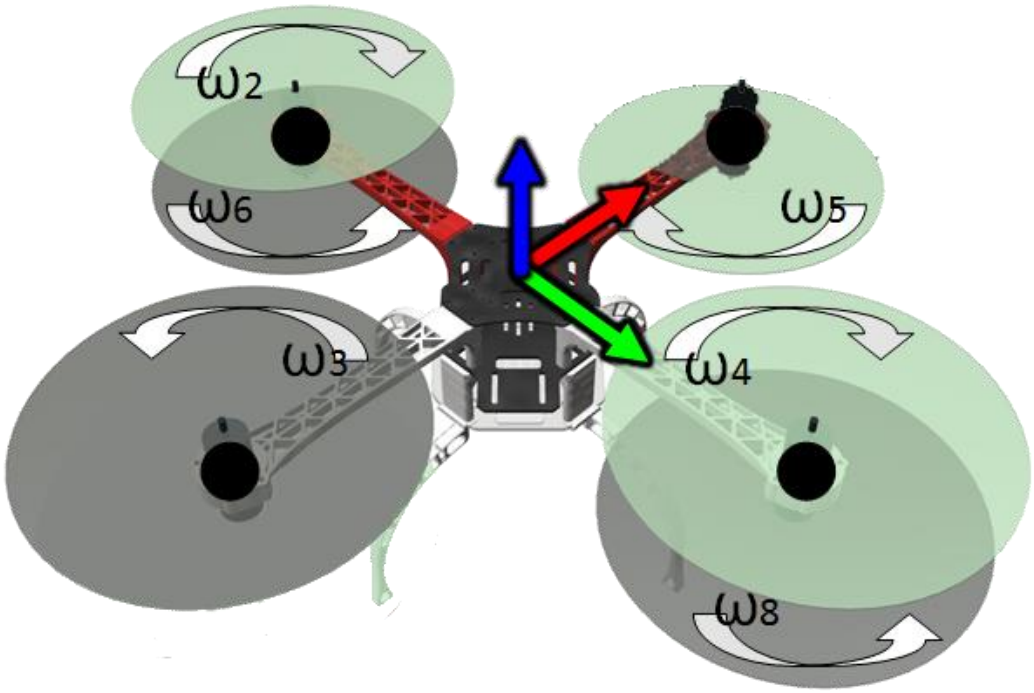


Figure 3.40 - Reconfiguration due to motor 1 failure.

With the understanding obtained from this study, one must conclude that only two motors can fail in the same axis being in the same side (upper or lower motors). In this case, all the motors from that side should be turned off to keep the balance of the yaw rotation (U_4). If motor 3 fails after motor one had already failed the result is the system of equations below.

$$\begin{cases} U_1 = f_2 + f_4 + f_5 + f_6 + f_7 + f_8 \\ U_2 = f_4 + f_8 - f_2 - f_6 \\ U_3 = f_7 - f_5 \\ U_4 = f_6 + f_8 - f_2 - f_4 - f_5 - f_7 \end{cases} \quad (3.6.5)$$

From the system of equations (3.6.5), one must observe that the force related with the yaw rotation becomes unbalanced. The solution to the problem is to shut down the motors from the same side (f_2 and f_4) and turn on the one that was shut down previously (f_7). These reconfigurations' results are expressed in the system of equation (3.6.6) which can be observed in Figure 3.41.

$$\begin{cases} U_1 = f_5 + f_6 + f_7 + f_8 \\ U_2 = f_8 - f_6 \\ U_3 = f_7 - f_5 \\ U_4 = f_6 + f_8 - f_5 - f_7 \end{cases} \tag{3.6.6}$$

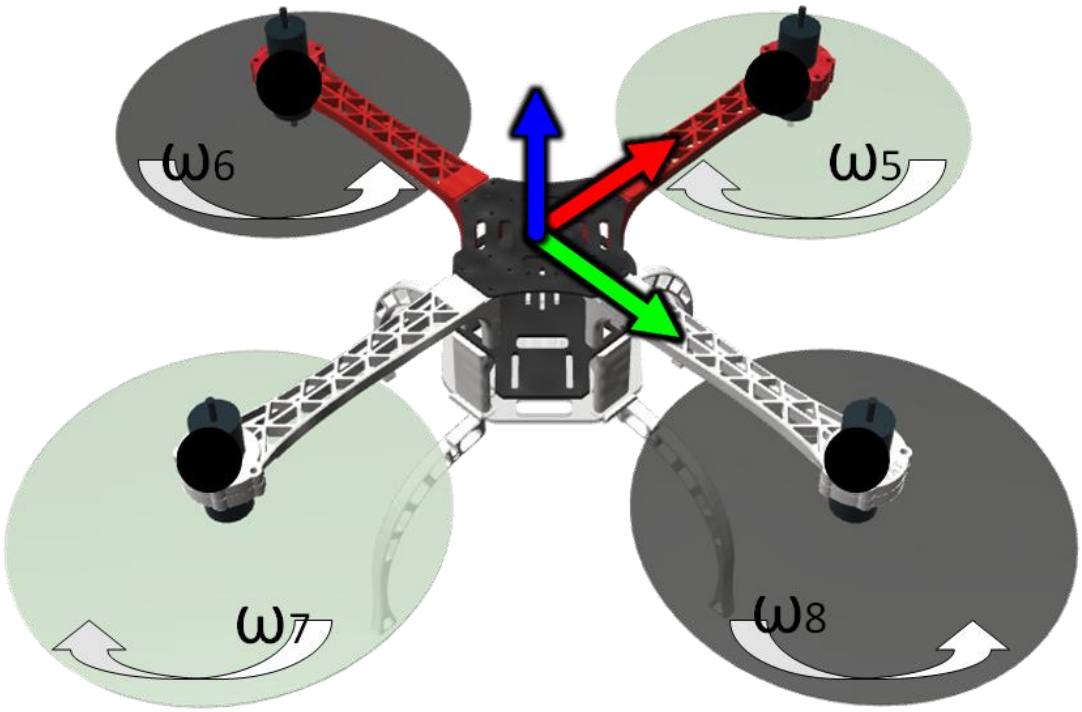


Figure 3.41 – Reconfiguration due to two motors failure (f_1 and f_3).

This last example lead the system architecture to a conventional four motors quadcopter. From this point no more actuators faults are tolerated.

Although the architecture allows applying system reconfiguration, the controllers need to be adjusted according to which motor failed. In order to select the controller adequate to the situation, some failures were considered. Pitch, roll and yaw were re-tuned according to the fault in hands.

Table 3.17 shows which motors should be deactivated according to each fault. In case of more than two motor failures, if they are on the same side (upper or lower) the result is similar to the two motor failures.

Table 3.17 - Motors failure and reconfigurations

Motor failure	*Reconfiguration*[4]	*Controller re-tuning*
Motor 1	Motor 7	Pitch and Yaw
Motor 2	Motor 8	Roll and Yaw
Motor 3	Motor 5	Pitch and Yaw
Motor 4	Motor 6	Roll and Yaw
Motors 1 and 3	Motors 2 and 4	Pitch, Roll and Yaw
Motors 2 and 4	Motors 1 and 3	Pitch, Roll and Yaw
Motors 5 and 7	Motors 6 and 8	Pitch, Roll and Yaw
Motors 6 and 8	Motors 5 and 7	Pitch, Roll and Yaw

The controllers' gains were obtained using the same methods described in the previous subsection (3.5.3). The redundancy of them along the fault simulation lead to a controller with fixed gains represented in Table 3.18.

[4] Motors to shut down.

Table 3.18 - Faulty PID gains.

Variable	K_p	T_i	T_d
Picth	0.05	4	5
Roll	0.05	4	5
Yaw	0.4	5	3.3

3.7 Quadcopter X8-VB Control Application

The design and implementation in the real quadcopter now takes place with the description of the algorithms used to fuse the sensors and the control algorithm implemented into the Arduino. Transforming sensor's data into information is presented first, the control algorithm implemented comes next and, later, the data logging and communications are explained.

3.7.1 Sensors

The autonomous flight of the aircraft demands several sensors and actuators. To transform the data from the sensors into information, some algorithms must take place.

First, the sensors have an initialization process in order to transmit their information to the microcontroller. This initialization can be observed in the attachment Arduino Code more precisely in the Initialization topic. Next, the sensors' data used in the work is described.

3.7.1.1 Absolute Orientation Sensor

The data provided by the Absolute Orientation Sensor is somehow treated by its microcontroller, in order to provide the Euler angles. Because of its microcontroller, no filter was needed for this sensor, as the steady state response from the sensor had no noise. This can be observer in Figure 3.42 and Figure 3.43.

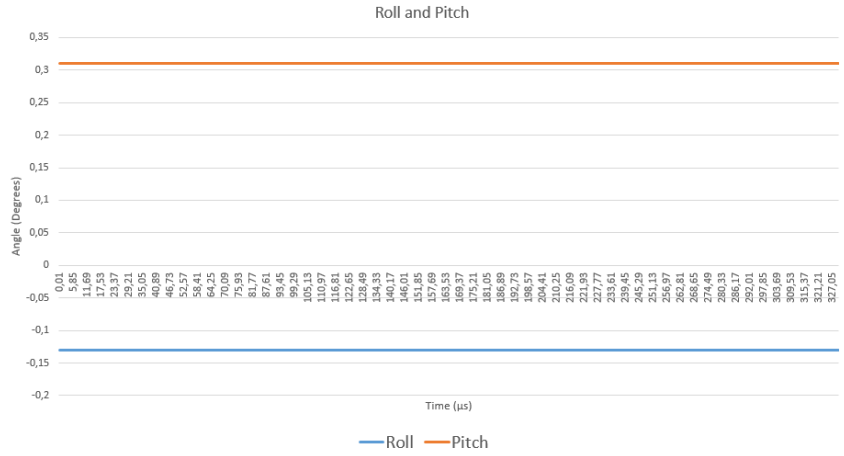


Figure 3.42 - Absolute Orientation Sensor pitch and roll steady state data experiment.

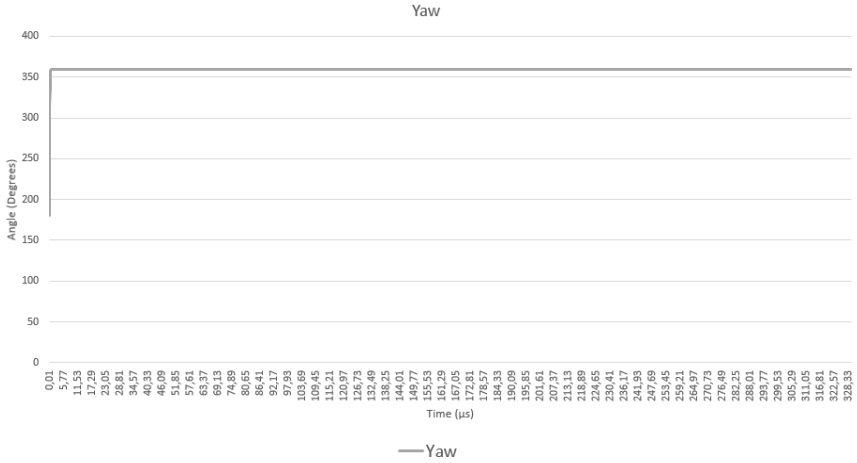


Figure 3.43 - Absolute Orientation Sensor yaw steady state data experiment

3.7.1.2 AltIMU

The AltiIMU from Pololu presented some steady state noise as the Figure 3.44 demonstrate.

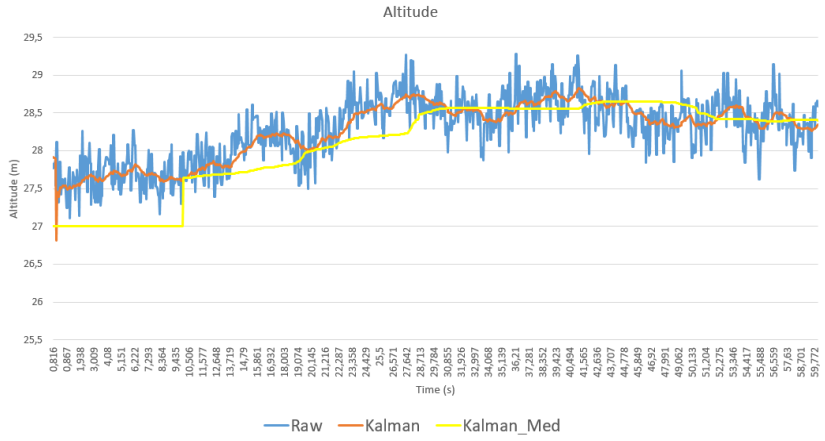


Figure 3.44 - AltIMU steady state data experiment.

Two filters were applied in order to smooth the sensor data. Kalman filter (Welch & Bishop, 2006) was the first to be applied and then a median filter of the past 10s of the experiment was implemented. The response of the kalman filter and the kalman plus the median filter can also be observed in Figure 3.44 where a better filter data is present.

3.7.1.3 ACS712 Current Sensor

The steady state response of the sensors is represented in Figure 3.45 where the analog input suffer an alteration in order to be translated to current values.

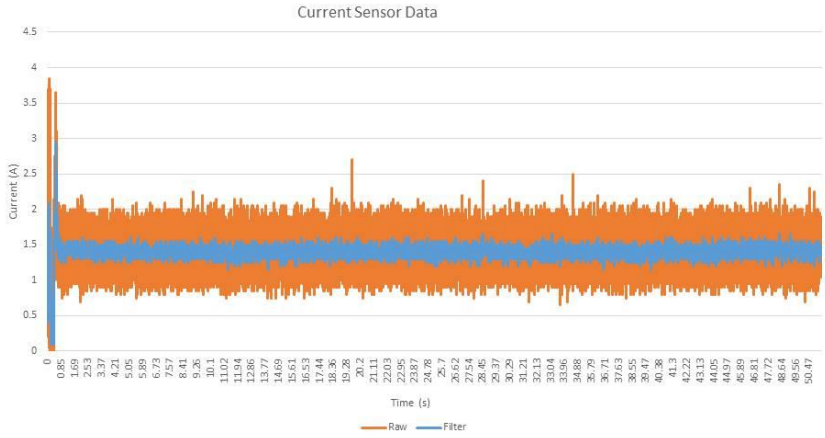


Figure 3.45 - Current sensor steady state response.

The sensor presented some variance but the signal noise ratio is less than one. A median filter was applied to reduce the variance.

3.7.2 Actuators

The controllers implemented in Arduino manipulated the actuators. The ESC have an initialization process and are manipulated with a PWM signal generated by a command provided by an Arduino library. Figure 3.46 shows the initialization function.

```
void initESC() {
  m1.u.attach(m1.pin); m1.u.writeMicroseconds(1100); m2.u.attach(m2.pin); m2.u.writeMicroseconds(1100);
  m3.u.attach(m3.pin); m3.u.writeMicroseconds(1100); m4.u.attach(m4.pin); m4.u.writeMicroseconds(1100);
  m5.u.attach(m5.pin); m5.u.writeMicroseconds(1100); m6.u.attach(m6.pin); m6.u.writeMicroseconds(1100);
  m7.u.attach(m7.pin); m7.u.writeMicroseconds(1100); m8.u.attach(m8.pin); m8.u.writeMicroseconds(1100);
  delay(1000);
}
```

Figure 3.46 – ESC initialization function code.

After the initialization, the motors are ready to receive the input from the controllers; this is done with an update function, which can be observed in Figure 3.47. The inputs of each motor follows equations (3.5.1).

```
void updateESC() {
  m1.u.writeMicroseconds((Throttle.input - ctrpitch.u + ctryaw.u)*F[0]); m2.u.writeMicroseconds((Throttle.input - ctrroll.u - ctryaw.u)*F[0]);
  m3.u.writeMicroseconds((Throttle.input + ctrpitch.u + ctryaw.u)*F[0]); m4.u.writeMicroseconds((Throttle.input + ctrroll.u - ctryaw.u)*F[0]);
  m5.u.writeMicroseconds((Throttle.input - ctrpitch.u - ctryaw.u)*F[0]); m6.u.writeMicroseconds((Throttle.input - ctrroll.u + ctryaw.u)*F[0]);
  m7.u.writeMicroseconds((Throttle.input + ctrpitch.u - ctryaw.u)*F[0]); m8.u.writeMicroseconds((Throttle.input + ctrroll.u - ctryaw.u)*F[0]);
}
```

Figure 3.47 – Example of ESC update function code.

3.7.3 Control Algorithms

3.7.3.1 Discrete PID controller

The PID controller implemented follows the same expressions as the one already explained in a previous section (A. PID controller). An example of the implementation for the pitch controller is presented in Figure 3.48.

```
void PID_pitch(float h) {

  ctrpitch.Tt = ctrpitch.ti;
  ctrpitch.bi = ctrpitch.kp * h / ctrpitch.ti;
  ctrpitch.ad = (2 * ctrpitch.td - ctrpitch.N * h) / (2 * ctrpitch.td + ctrpitch.N * h);
  ctrpitch.bd = 2 * ctrpitch.kp * ctrpitch.N * ctrpitch.td / (2 * ctrpitch.td + ctrpitch.N * h);
  ctrpitch.ao = h / ctrpitch.Tt;

  ctrpitch.e = ctrpitch.sp - Sensors.pitch;
  ctrpitch.P = ctrpitch.kp * ctrpitch.e;
  ctrpitch.D = ctrpitch.ad * ctrpitch.D - ctrpitch.bd * (Sensors.pitch - ctrpitch.yold);
  ctrpitch.v = ctrpitch.P + ctrpitch.I + ctrpitch.D;

  if isnan(sat(ctrpitch.v, ctrpitch.ulow, ctrpitch.uhigh)) {
    Serial2.println("Controller NAN Error!");
    Serial.println("Controller NAN Error!");
    ctrpitch.u = 0;
    ctrpitch.I = 0;
  }
  else {
    ctrpitch.u = sat(ctrpitch.v, ctrpitch.ulow, ctrpitch.uhigh);
  }
  ctrpitch.I += ctrpitch.bi * ctrpitch.e + ctrpitch.ao * (ctrpitch.u - ctrpitch.v);
  ctrpitch.yold = Sensors.pitch;
}
```

Figure 3.48 - Example of discrete PID Arduino implementation.

Here the sampling time used was $h = 50ms$ since it corresponded to half of the algorithm computing time.

3.7.3.2 SMC controller

The SMC controller was already discretized and no modification was made to the algorithm presented previously (B. SMC controller). Figure 3.49 represents the implementation in the Arduino relative to the pitch rotation where $h = T_s$.

```
void SMC_Pitch(float h){

  smcpitch.e = smcpitch.sp - (Sensors.pitch / 20); //Normalization

  smcpitch.de = smcpitch.ge * (smcpitch.e - smcpitch.eold) / h + smcpitch.pc * smcpitch.deold;

  smcpitch.gaw = h / (smcpitch.c / smcpitch.lambda);

  smcpitch.ie = smcpitch.ieold + h * smcpitch.e + smcpitch.gaw * (smcpitch.uold - smcpitch.vold);

  smcpitch.oc = smcpitch.c * smcpitch.e + smcpitch.lambda * smcpitch.ie + smcpitch.de;

  smcpitch.v = smcpitch.rhoc * (smcpitch.oc / (abs(smcpitch.oc) + smcpitch.ec));

  smcpitch.u = sat(smcpitch.v, smcpitch.ulow, smcpitch.uhigh);

  smcpitch.eold = smcpitch.e;
  smcpitch.deold = smcpitch.de;
  smcpitch.ieold = smcpitch.ie;
  smcpitch.uold = smcpitch.u;
  smcpitch.vold = smcpitch.v;

}
```

Figure 3.49 – Example of SMC Arduino implementation.

3.7.4 Supervisor

The supervisor is always monitoring the motors' current (A) in order to detect any issue with them. It shuts down the motors according to the pre-set reconfiguration. The algorithm implemented can be observed in the attachments (V. Supervisor).

3.7.5 Communication

The communication algorithm is ready in the aircraft side; it sends the data from all sensors, control actuations and setpoints to the base station. The base station was not implemented with a user-friendly interface due to the lack of time. The commands are sent by the command line provided by the Arduino console. The data string starts with the character "$" followed by:

Current time (s) ; SP yaw ; actuation yaw ; sensor yaw ; SP roll ; actuation roll ; sensor roll ; SP pitch ; Actuation pitch ; sensor pitch ; SP Z ; actuation Z ; sensor Z ; Current on each motor

The algorithm implemented can be analyzed in the attachments section (III. Communications). The communication also provides information like motors fault, setpoint alteration among others.

3.7.6 Data Logging

The data logging intent is to save important information in case of communication loss. The data is saved in a .txt file in a micro-SD card, this way the data can be open by any user. It follows the same architecture of data presented before. The algorithm implemented can be observed in the attachments (IV. Data Logging).

Chapter 4

4 Simulation and Experimental Results

"The experiments help to understand the theory better." (Brito Palma, 2007).

Simulations and experimental results are the proof of the theory presented during Chapter 3. This section intent is to present the designs done before working with the model implemented and with the real drone.

4.1 Simulations

The simulation phase was the first to take place, in order to test the controllers and supervisors. It follows the same line of thought as before to easily understand the procedures.

4.1.1 PID Controllers

4.1.1.1 Simulations based on controllers tuned by Ziegler-Nichols and Åström methods

A. Altitude controller

Simulation of the switching PID controller to altitude can be observed in Figure 4.1. The results present good performance with practically no over-shoot and a maximized raising time.

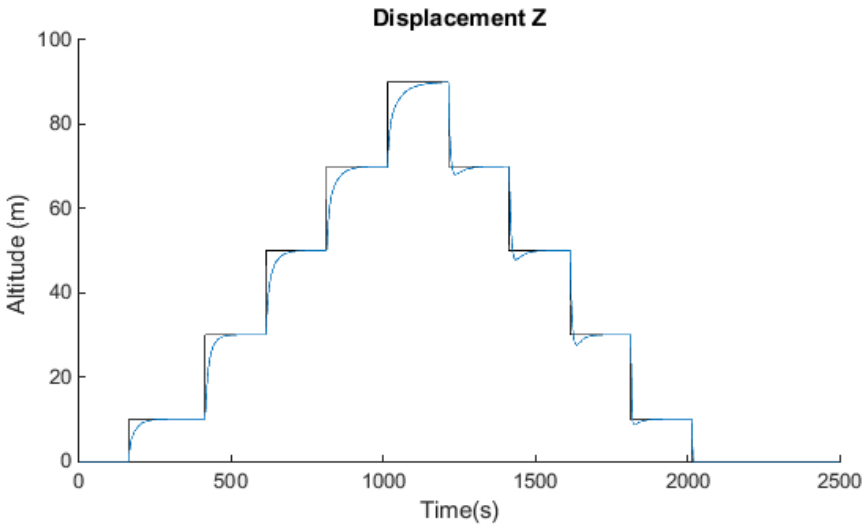


Figure 4.1 – Altitude switching PID control simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

Since the real implementation would not contemplate such range of altitude, simulation of the controller responsible for the low attitude (0; 20) was simulated again independently. Figure 4.2, Figure 4.3 and Figure 4.4 demonstrated the low altitude response of the PID controller.

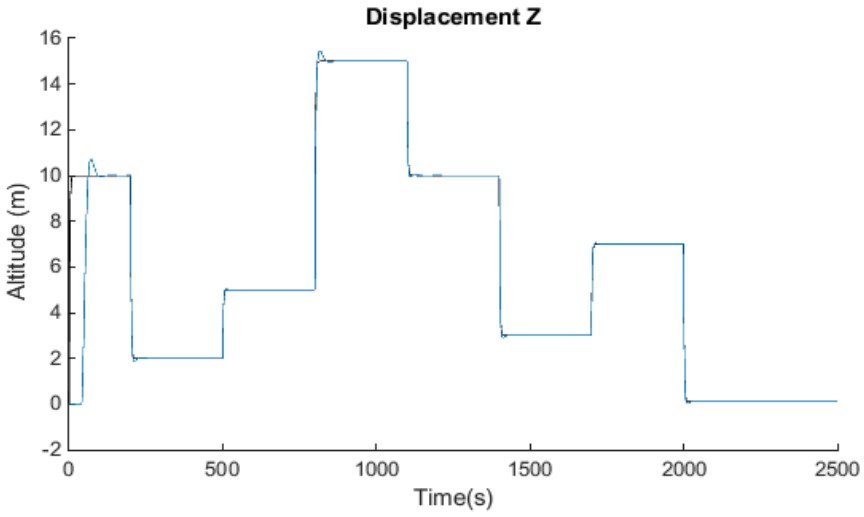


Figure 4.2 - Low altitude PID response simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

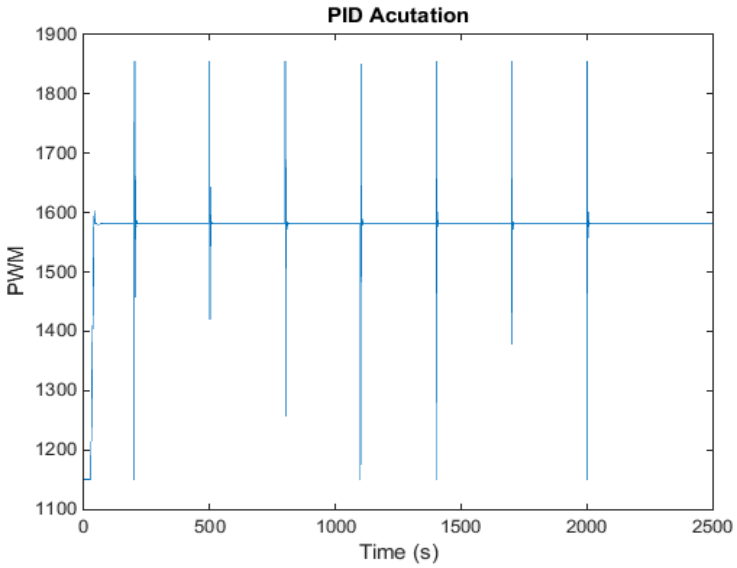


Figure 4.3 – Altitude PID actuation.

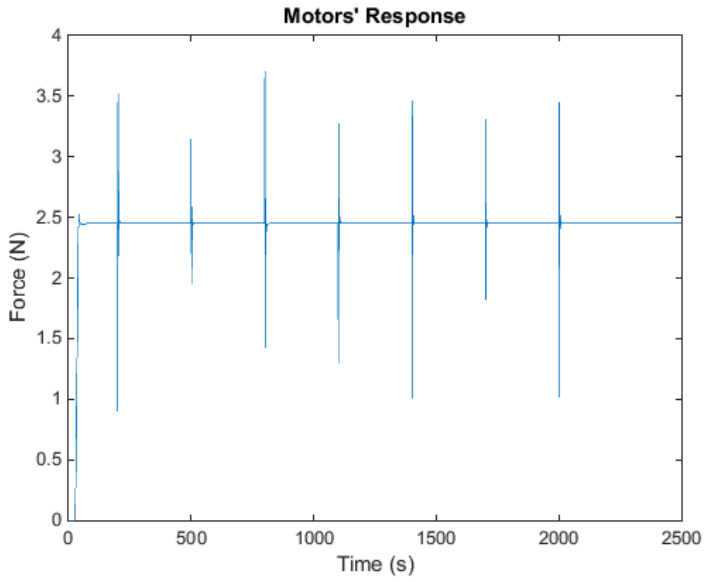


Figure 4.4 – Altitude motors' response to the PID actuation.

The PID actuation for this simulation can be observed in Figure 4.3. One can observe the hovering value of actuation and the spikes of actuation according to the setpoint variation. The motors response can be observed in Figure 4.4.

The response of each motor is directly related with the PID actuation value; this can be observed in the motors response. In conclusion, the PID controller for the altitude has a good performance with good actuation variance.

B. Pitch / Roll controllers

Pitch and Roll controllers have the same gains and similar behaviors. The simulations were first made individually and then simultaneously with the same setpoint and different setpoints. The individual simulations can be observed in Figure 4.5, Figure 4.6 and Figure 4.7.

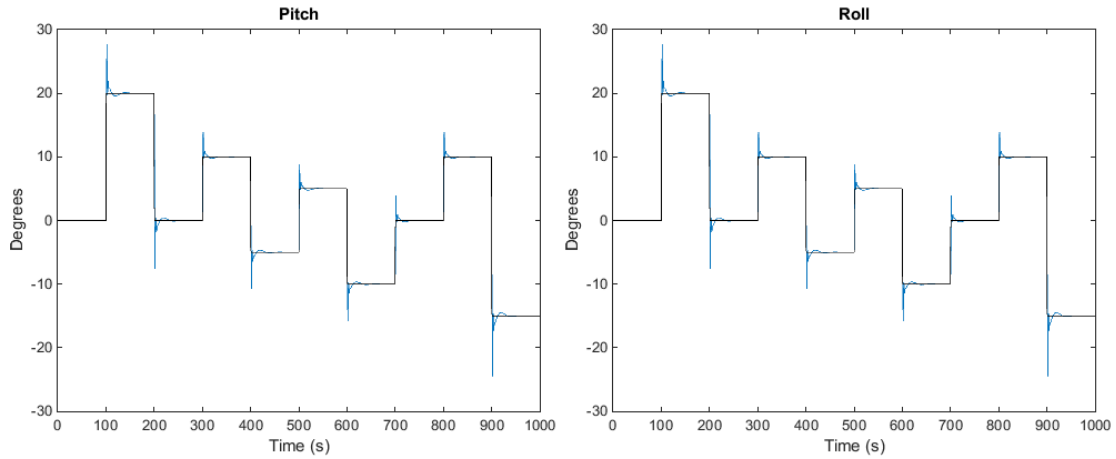


Figure 4.5 - Pitch and roll independent simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

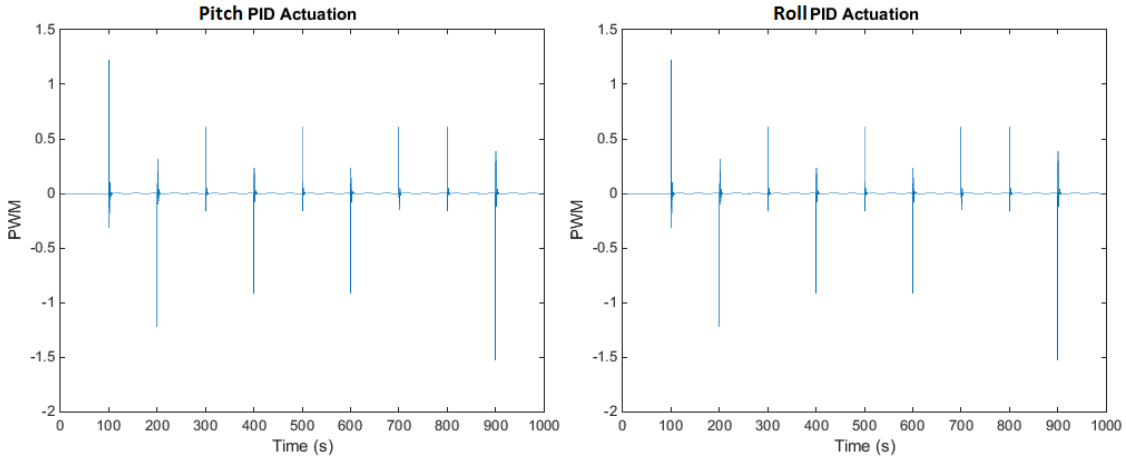


Figure 4.6 - PID actuation to pitch and roll.

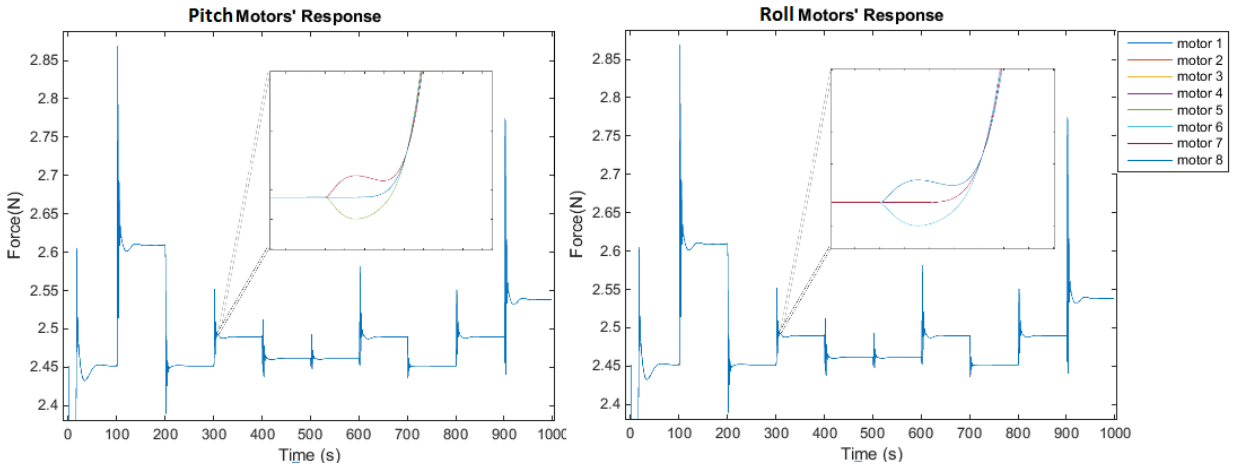


Figure 4.7 - Motors' response for pitch and roll simulations.

One must conclude from the simulations that the controller has good performance, although some overshoot is present on both rotations. This happens because the controller was adjusted to maximize the raising time, being the parameter selected to compare the model and the real world aircraft behaviors. In Figure 4.7 the hoovering force is manipulated and the control action from the altitude controller has some variations due to the loss of some lift from the rotations that has some reflects on the altitude. The zoom effectuated allows to observe the difference between the motors response due to their symmetric control input and the motors that do not have the corresponded control input.

Simulation with both rotations done at the same time with different setpoints was also made. The result of the closed-loop system's behavior can be analyzed in Figure 4.8.

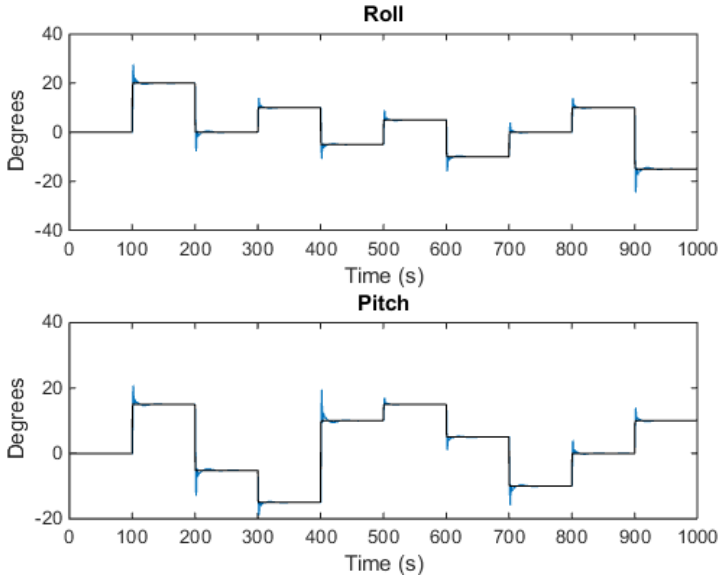


Figure 4.8 – Pitch and roll simultaneous simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

The result from the previous simulation presented good performance, with both rotations following the setpoint with minimal error and the intended raising time.

C. Yaw controller

The yaw controller was also simulated independently. The three rotations were not simulated simultaneously, since the trajectory controllers are applied first to yaw. After the yaw rotation has minimal error, trajectory controllers are applied to the roll and pitch rotations. This is done since the influence of the yaw rotation to the motors leads the closed-loop system to instability when pitch and/or roll rotations are changing simultaneously with yaw. Observe the simulation of the PID that controls the yaw rotations in Figure 4.9, Figure 4.10 and Figure 4.11.

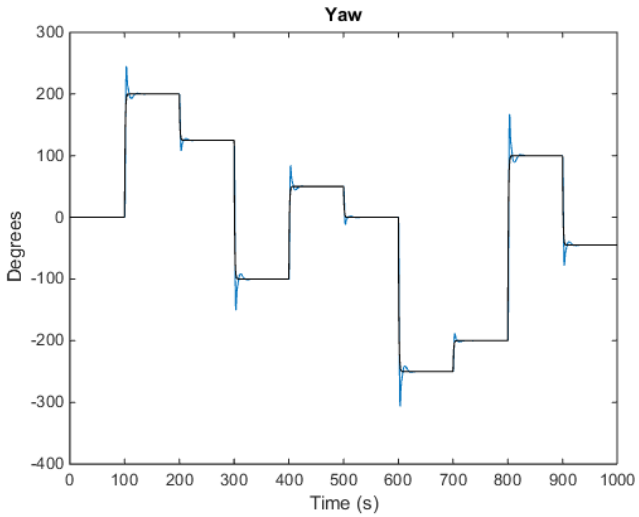


Figure 4.9 - PID Yaw simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

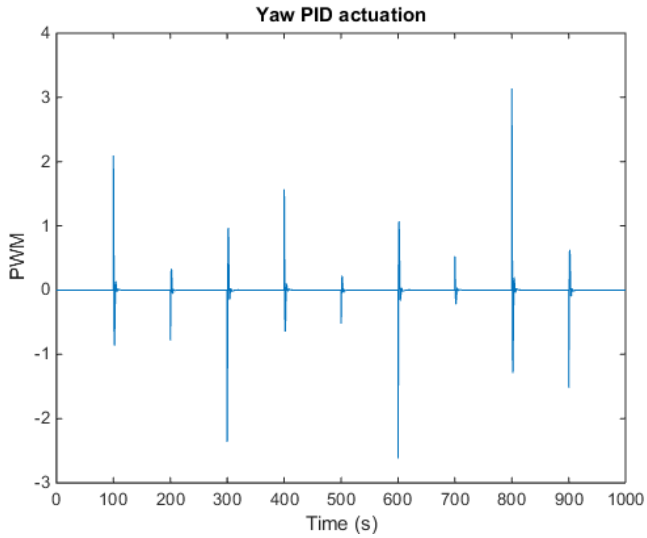


Figure 4.10 - PID actuation for yaw rotation.

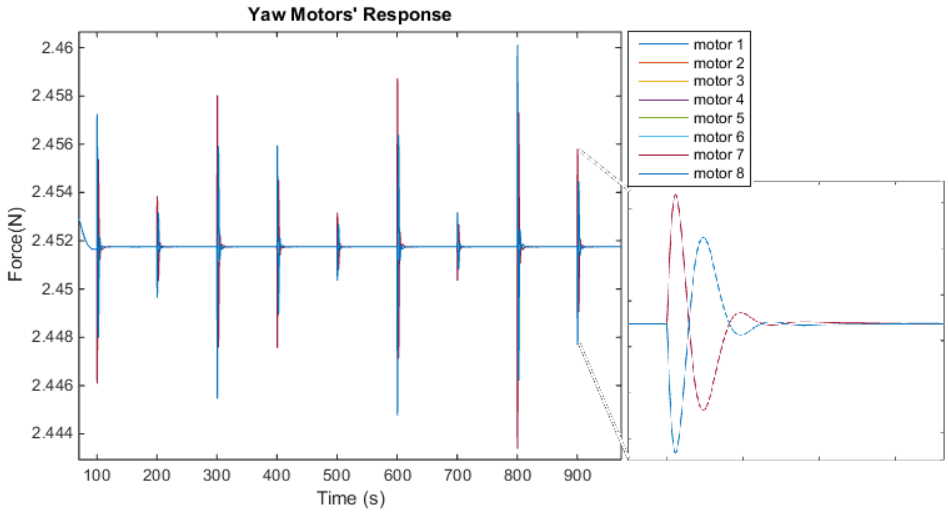


Figure 4.11 - Motors' response for yaw simulation.

The controller presented good performance, since the raising time was the one optimized. Due to this, some overshoot is present in each setpoint alteration. A filter to the setpoint was later applied in order to reduce the overshoot.

The control actuation variance is low, as intended, and one must conclude from observing Figure 4.11 that the hoovering force was not altered while the yaw position was different from zero; this was expected since the yaw controller actuation is applied to all motors.

D. Position and Trajectory controllers

The position and trajectory controllers were tested considering both methods mentioned before – with yaw and without yaw rotation. The first simulations were made without yaw rotation, because it is the simplest. The displacements were first simulated individually and then at the same time. Figure 4.12 and Figure 4.13 demonstrate the individual simulations; Figure 4.14 and Figure 4.15 demonstrate the simultaneous simulation result.

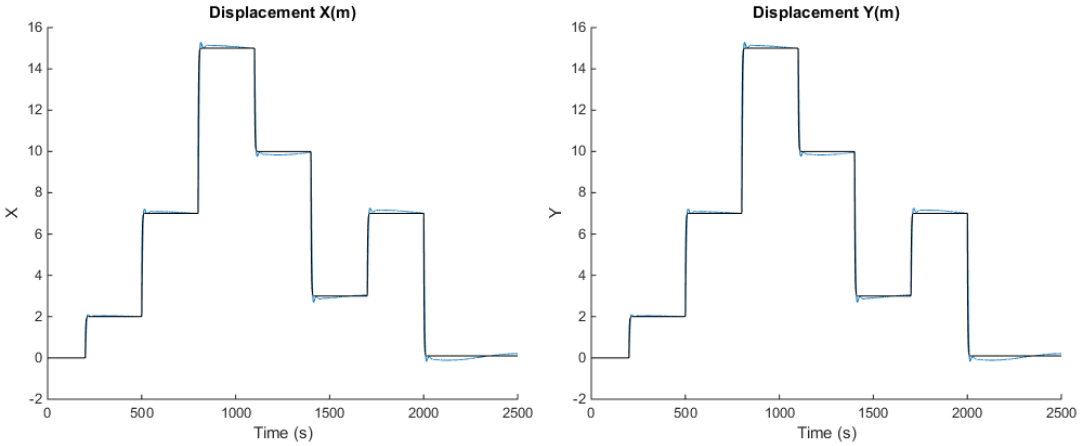


Figure 4.12 – PID X and Y independent simulations. The black line represents the setpoint and the blue line the closed-loop system's response.

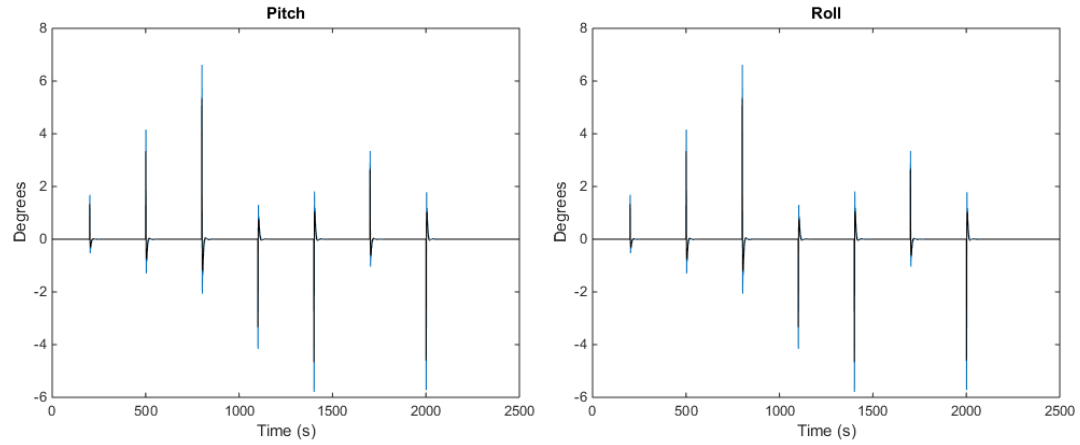


Figure 4.13 – Position PID actuation and rotations' response. The black line represents the setpoint and the blue line the closed-loop system's response.

The independent PID controllers' simulations presented good performance, as well as the behavior of the correspondent rotations. With these results, simulation with both controllers simultaneously took place.

I. Position and trajectory controllers without yaw rotation

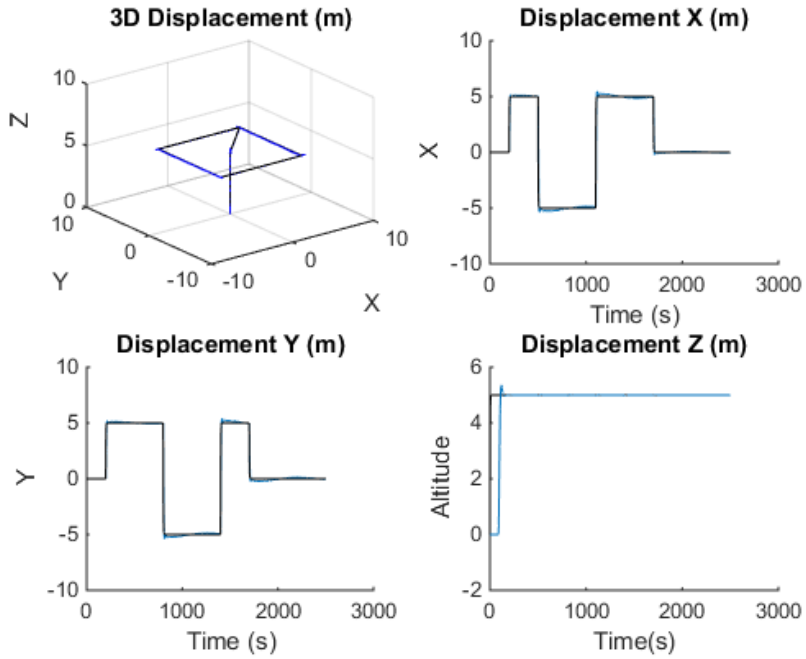


Figure 4.14 - Trajectory PID controllers without yaw rotation. The black line represents the setpoint and the blue line the closed-loop system's response.

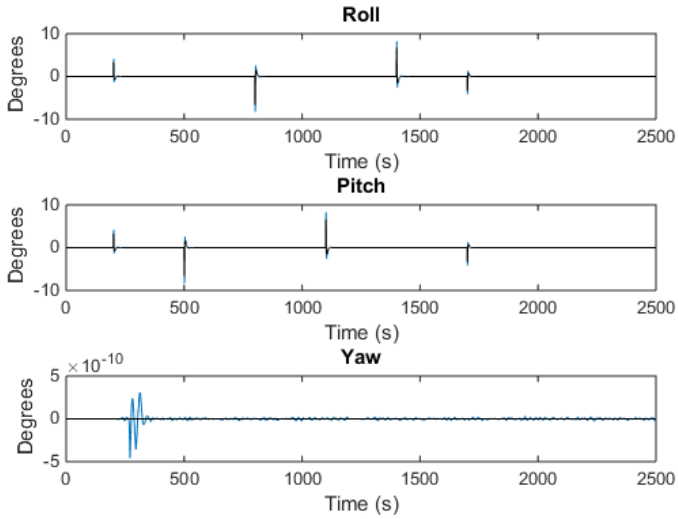


Figure 4.15 - Trajectory PID controllers actuations and rotations response. The black line represents the setpoint and the blue line the closed-loop system's response.

The PID controllers presented good performance with expected raising time and low overshoot. The position was maintained with minimal error and the trajectory was also made with good performance.

The influence of pitch and roll rotation to yaw can be observed in Figure 4.15. The influence is very low but it is present during the entire simulation. The rotation response presented good performance to the variable setpoint, fulfilling the requirements to accomplish the intended trajectory.

II. Position and trajectory controllers with yaw rotation

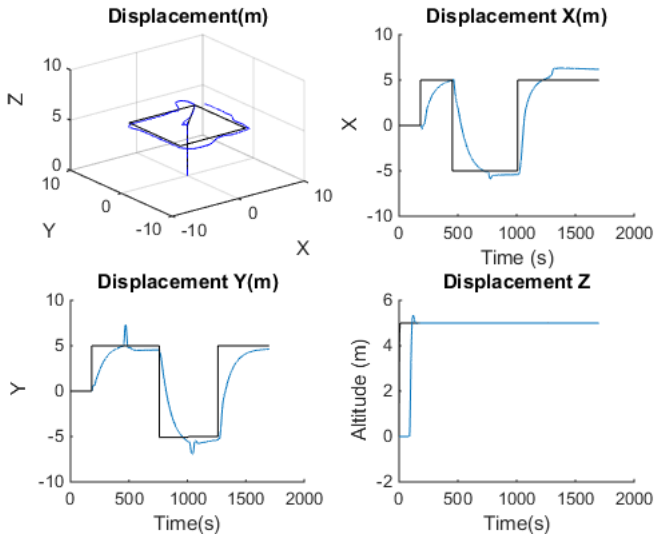


Figure 4.16 - Trajectory PID controllers with yaw rotation. The black line represents the setpoint and the blue line the closed-loop system's response.

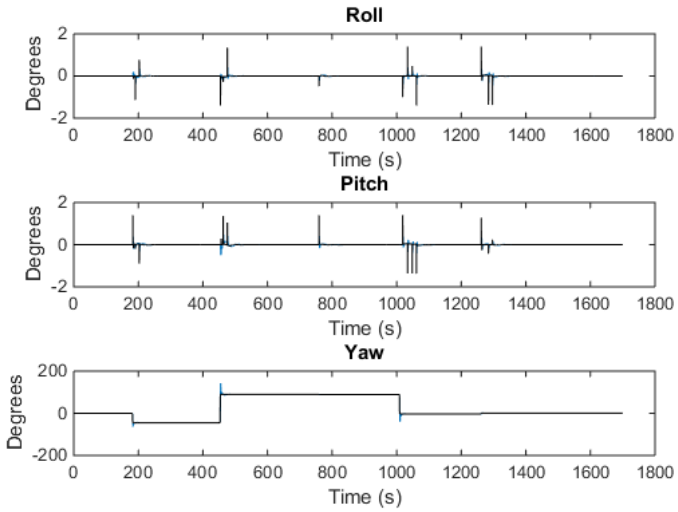


Figure 4.17 - Trajectory PID controllers actuations and rotations response. The black line represents the setpoint and the blue line the closed-loop system's response.

The trajectory with yaw rotation is a more complex matter then without it. Some simulations were made as the one from Figure 4.16 and Figure 4.17. The performance of the trajectory was not the intended, aside from completing it.

The coupling of the yaw rotations and the transformation of the coordinates mentioned before adds some perturbations to the position and trajectory controllers. This can be observed by the trajectory made, since it did not follow the setpoint line. However, the position was fulfilled because the vertices, which are the waypoints, were reached with success.

The methods of position and trajectory presented are just in an early stage being proposed to future work, in order to maximize the performance of the trajectories. Next, controller obtained from the PSO simulations will be presented.

4.1.1.2 Simulations based on PSO

The procedure was to simulate the yaw rotation separately from the other components. This procedure was made since trajectory with yaw was not the intended objective.

A. Yaw controller

The result from the simulation can be observed in Figure 4.18 and Figure 4.19. The rotation normalization are from [-1; 1] which represent displacement between [-180; 180] (degrees).

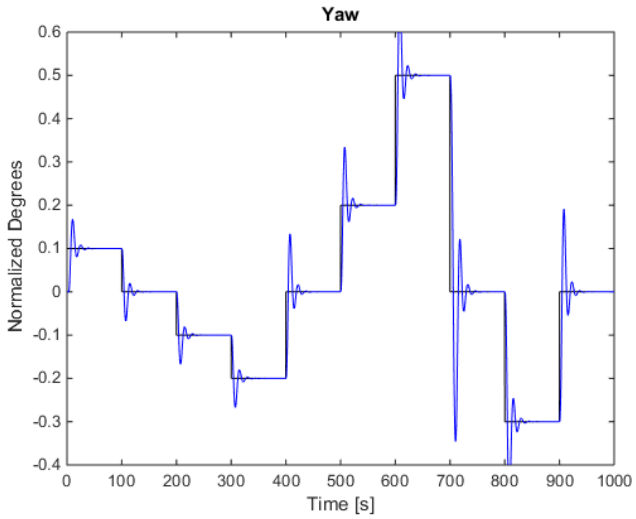


Figure 4.18 – Yaw PID, based on PSO, simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

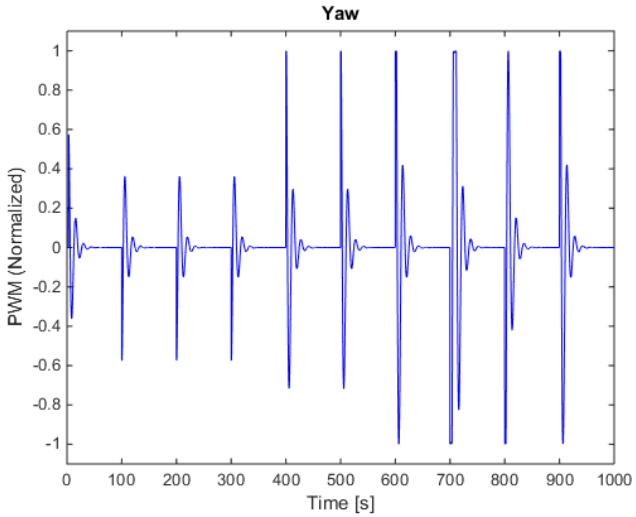


Figure 4.19 – PID, based on PSO, actuation for yaw rotation.

The result offered a sub-optimal solution and the control based on the Ziegler-Nichols rules presented better performance. This could be even more optimized with more particles and more iterations for each particle. This is a matter of enlarging the optimization time in order to obtain a better controller solution.

B. Position and trajectory controller without yaw rotation

Position and trajectory controllers without yaw rotation were tested with the gains obtained from the PSO. The results from the simulations can be observed in Figure 4.20 and Figure 4.21. The positions' normalization is from [-1; 1] which represents displacement between [-20; 20] (m) and the rotations' normalization is from [-1; 1] which represents angles between [-20; 20] (degrees).

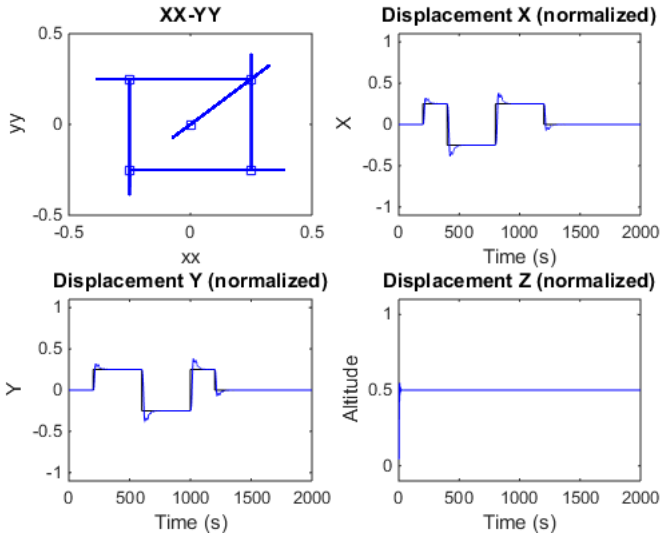


Figure 4.20 – PSO PID controllers without yaw rotation applied to trajectory. The black line represents the setpoint and the blue line the closed-loop system's response.

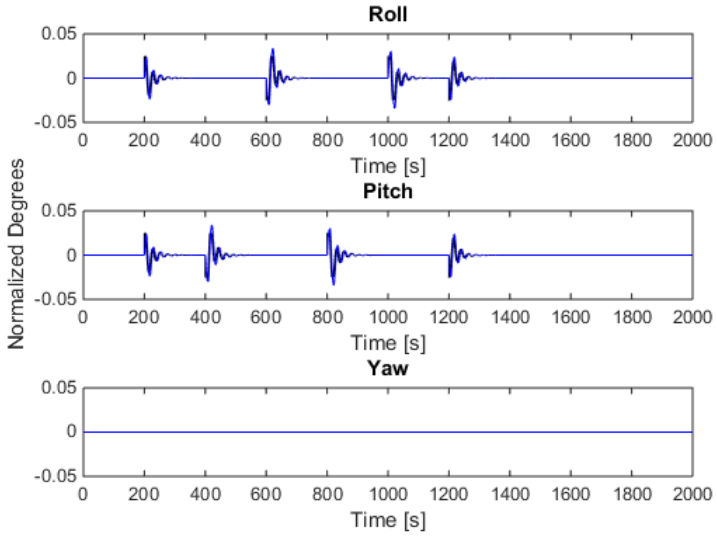


Figure 4.21 - PSO PID controllers' actuations and rotations response. The black line represents the setpoint and the blue line the closed-loop system's response.

The results obtained from the controllers acquired from the PSO algorithm demonstrated similar results to the controllers designed previously. As mentioned before, the optimization process requires some time and powerful hardware in order to achieve better solutions.

4.1.2 SMC Controller

We followed the same procedure for the SMC controller as in the PID controller with the yaw rotation being simulated separately and the position and trajectory being simulated later.

A. Yaw controller

Observe the result from the simulation of the yaw rotation with the SMC controller in Figure 4.22 and Figure 4.23.

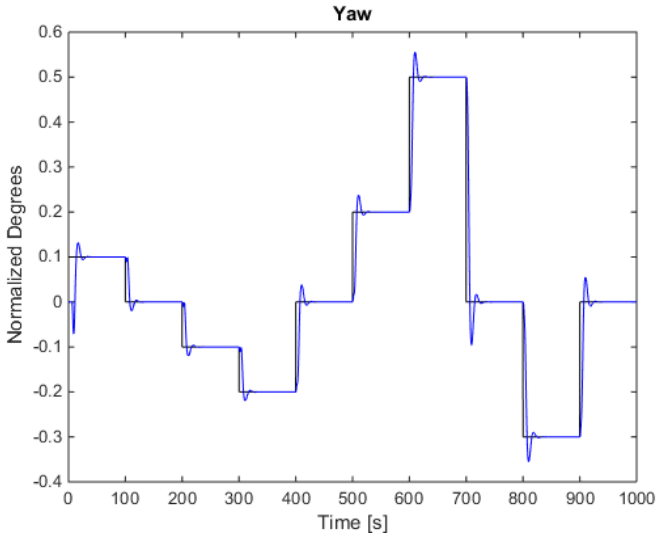


Figure 4.22 - Yaw SMC, based on PSO, simulation. The black line represents the setpoint and the blue line the closed-loop system's response.

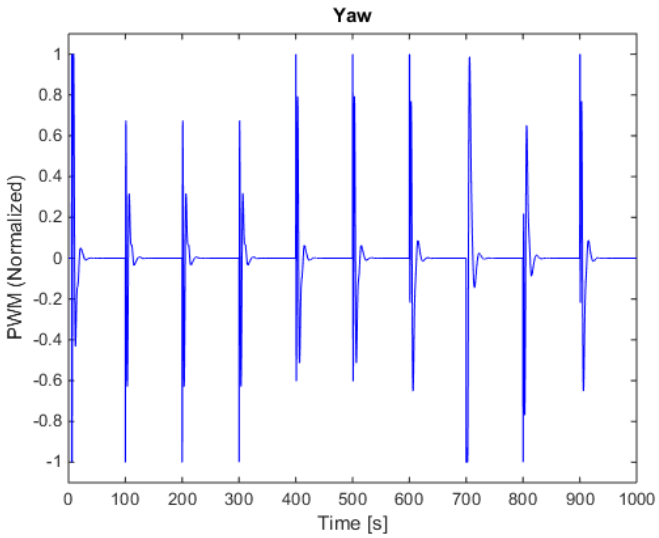


Figure 4.23 - SMC, based on PSO, actuation for yaw rotation.

The SMC controller applied to the yaw rotation presented good performance with less overshoot and control variance than the PID controller. The result is as usual a sub-optimal solution.

B. Position and trajectory without yaw rotation

The SMC controller tuned with PSO algorithm was tested with similar setpoint as the PID, the result is presented in Figure 4.24 and Figure 4.25.

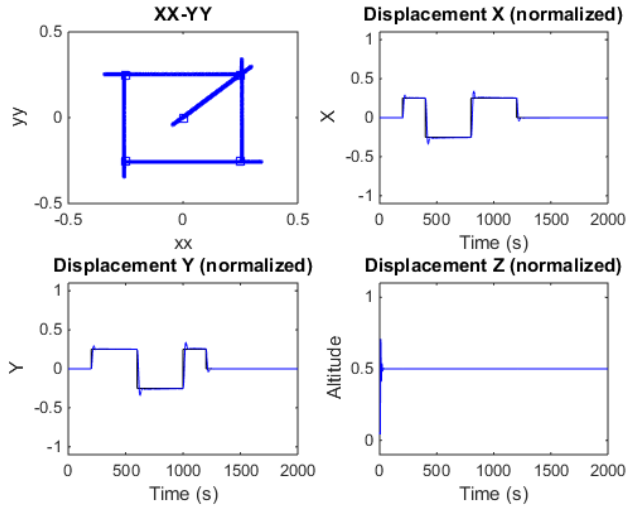


Figure 4.24 - PSO SMC controllers without yaw rotation applied to trajectory. The black line represents the setpoint and the blue line the closed-loop system's response.

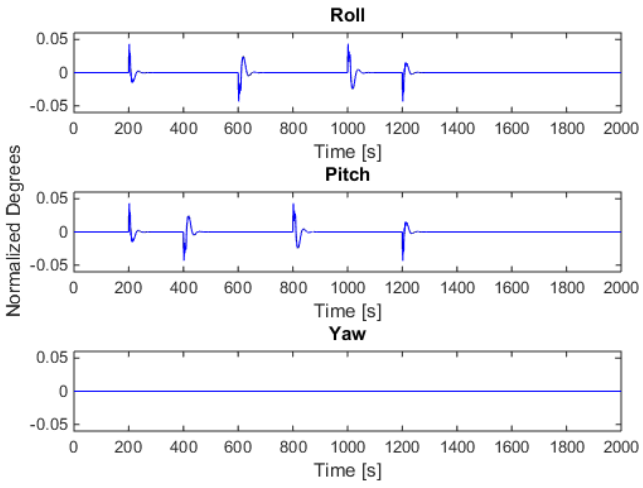


Figure 4.25 - PSO SMC controllers' actuations and rotations response. The black line represents the setpoint and the blue line the closed-loop system's response.

The SMC controller applied to trajectory presented good performance. Similarly to the yaw rotation, the SMC controller presented less overshoot and control variance than the PID. The better performance of the SMC is because it is based on a non-linear design, on the other hand, the PID controller is a linear design.

4.1.3 Fault Tolerant Control

Fault tolerance and reconfiguration was simulated, in order to better understand the closed-loop system's behavior described in the previous chapter.

Simulations were made to several faults and their correspondent reconfigurations all presenting good performance. In order to illustrate one of the fault tolerance and reconfiguration, a simulation was made considering the fault of only one motor. To better comprehend the example made in the Fault Tolerant Control studies, the simulation will use the fault of the same motor (motor 1).

The simulations have two components: one with the base station supervisor and another with the on-board supervisor. The difference is that, with the latter, the aircrafts lands immediately and, with the first, the drone maintains its flight.

First the simulations were made considering the conection to the base station supervisor. This means that a new set of controllers' gains go into action, in order to conpensate the fault and ajust to the refonfiguration, also changing the allocation of control.

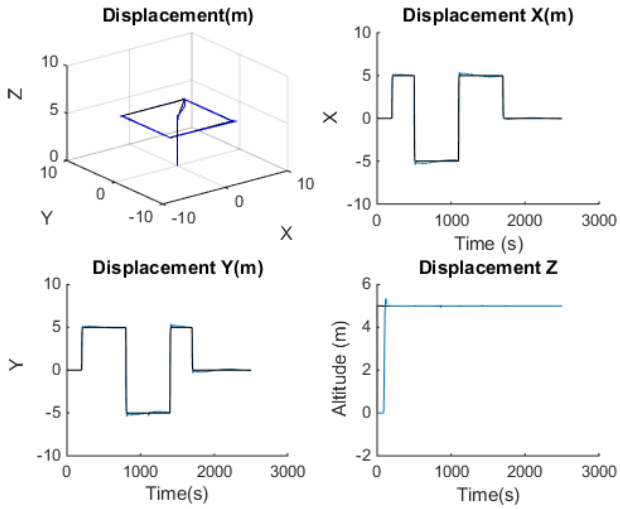


Figure 4.26 - Position and trajectory tolerant fault control to motor 1 failure. The black line represents the setpoint and the blue line the closed-loop system's response.

The Figure 4.26, Figure 4.27 and Figure 4.28 show the reconfiguration and re-tuning due to motor 1 failure at 850s. One must observe that, from the time that the motor failed, the yaw rotation has some perturbation each time the pitch is manipulated. A slight change is also observed in the altitude with the loss of some lift.

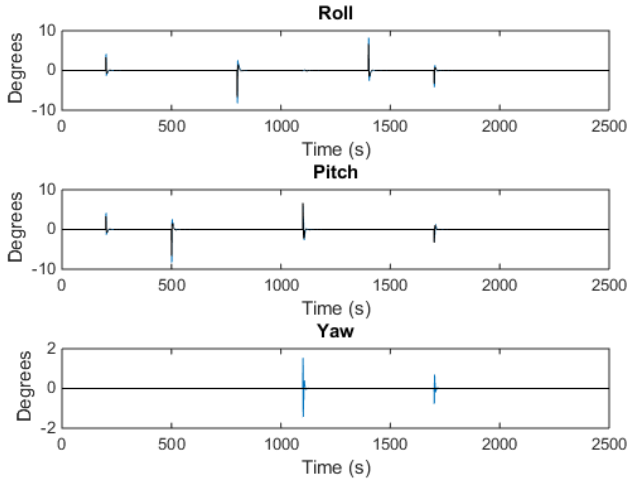


Figure 4.27 - Position and trajectory rotations fault tolerant control to motor 1 failure. The black line represents the setpoint and the blue line the closed-loop system's response.

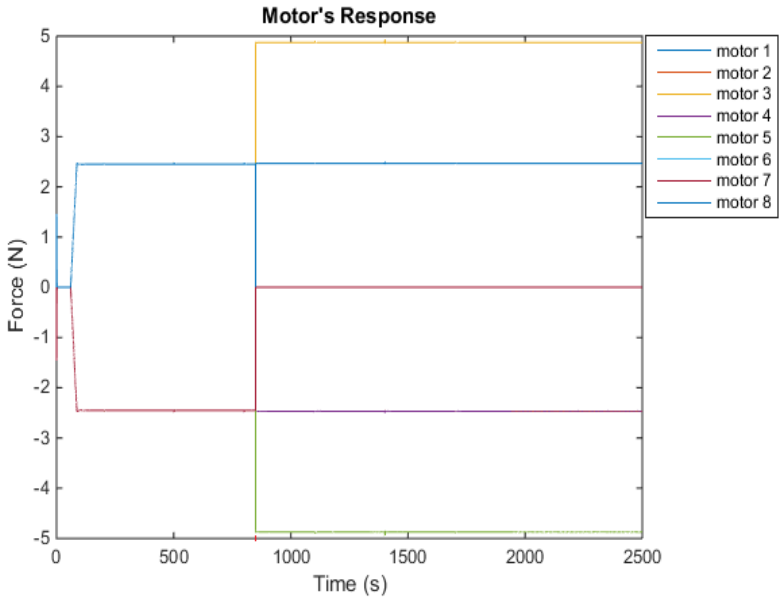


Figure 4.28 - Position and trajectory motors' response to motor 1 failure. The negative values represent the motors rotating in the opposite direction in relation the yaw positive rotation.

Aside the actuator fault, the objective of fault tolerance is obtained with the fulfillment of the trajectory with as good performance as a non-faulty operation mode. The second situation, where the connection to the base station supervisor is lost, was simulated presenting the results in the following figures (Figure 4.29, Figure 4.30 and Figure 4.31).

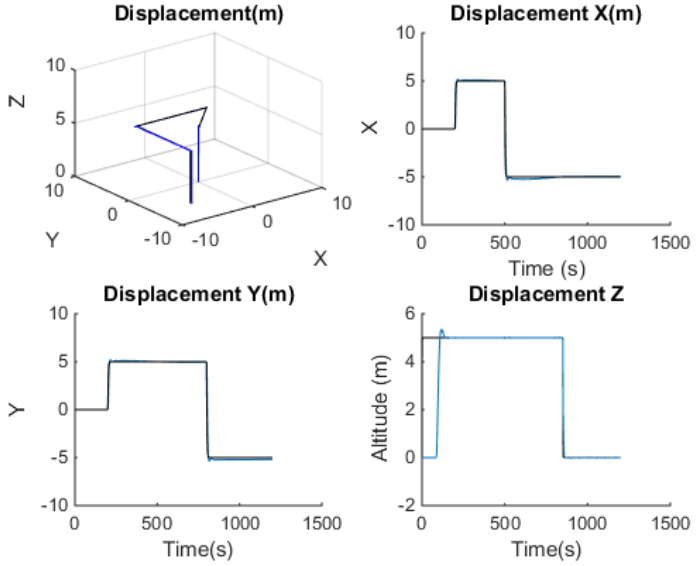


Figure 4.29 - Position and trajectory fault tolerant control to motor 1 failure, safe landing. The black line represents the setpoint and the blue line the closed-loop system's response.

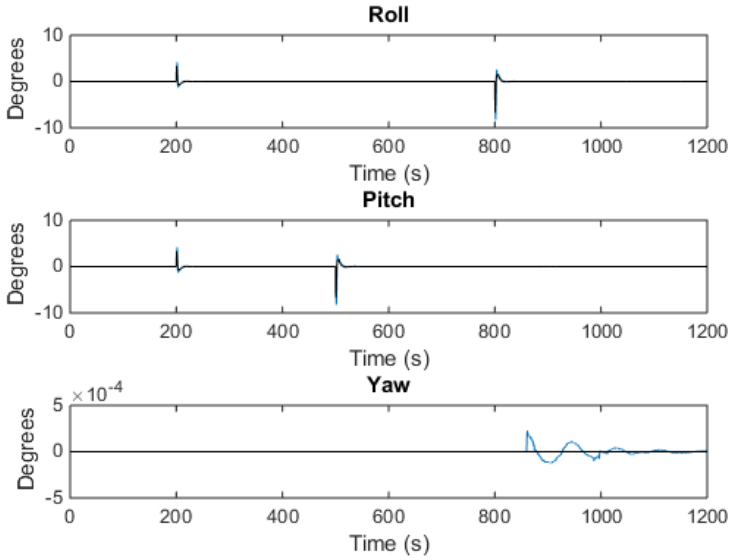


Figure 4.30 - Position and trajectory rotations fault tolerant control to motor 1 failure, safe landing. The black line represents the setpoint and the blue line the closed-loop system's response.

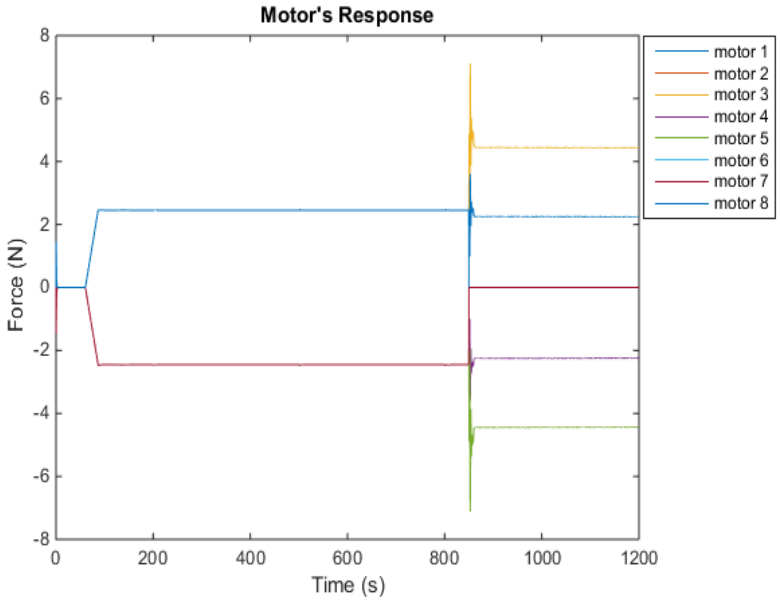


Figure 4.31 - Position and trajectory motors' response to motor 1 failure. The negative values represent the motors rotating in the opposite direction in relation the yaw positive rotation.

4.2 Experimental Results

Experimental results were not made with the same sequence as the simulations. For the real trials, the attitude of the aircraft was the one tested. The aircraft was restrained with steel cables, in order to test the attitude controllers and the altitude controller. Figure 4.32 shows one of the restrained configurations.

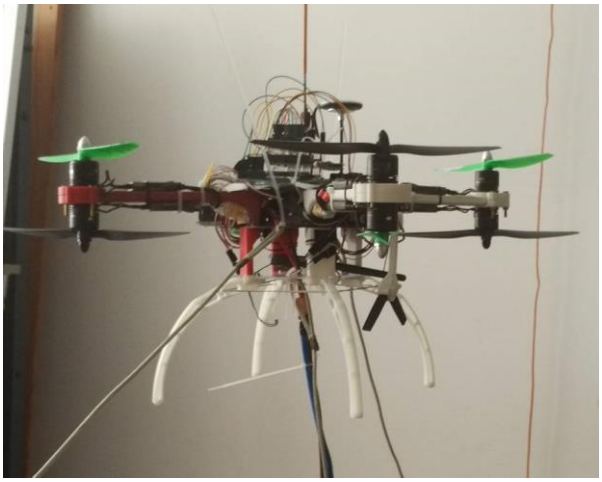


Figure 4.32 - Pitch experiment restrain configuration.

4.2.1 PID Controller

4.2.1.1 Pitch / Roll controller

Pitch/Roll were the first to be tested. The test was made with a permanent perturbation due to the cables that were holding the drone; this was a good way to test the controller. Figure 4.33, Figure 4.34, Figure 4.35 and Figure 4.36 represent the data received by the base station containing the sensor data and the controller actuation.

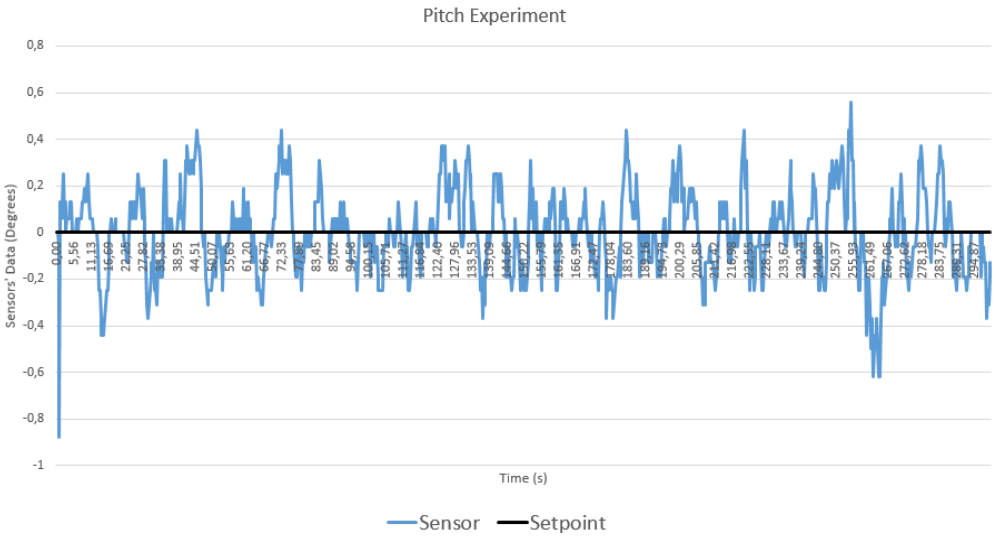


Figure 4.33 – Pitch sensor data received by the base station.

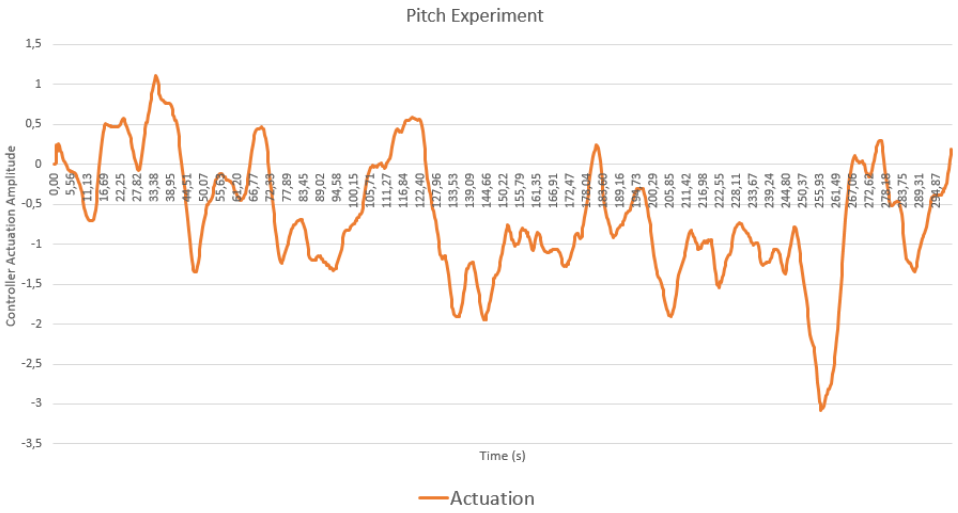


Figure 4.34 – Pitch PID control actuation.

The pitch was tested with a permanent perturbation of 0.5º. The roll, on the other hand, was tested with less permanent perturbation around -1.5º. These permanent perturbations are reflected in the offset of the controllers' actuation.

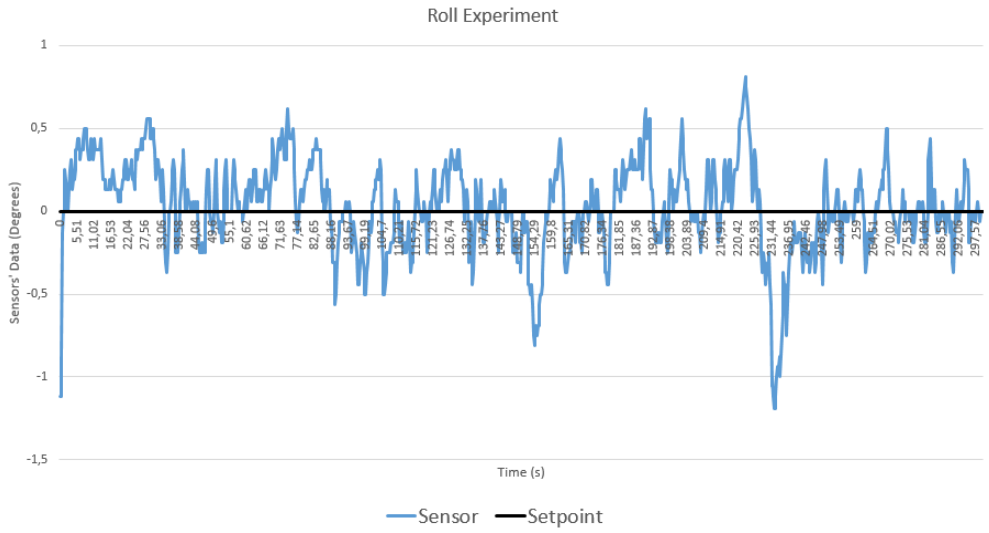


Figure 4.35 - Roll sensor data received by the base station.

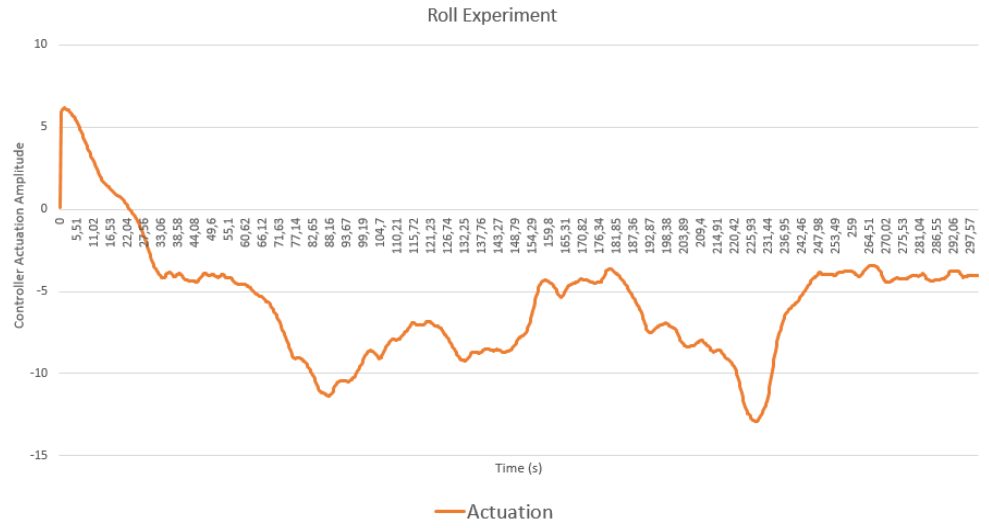


Figure 4.36 - Roll PID control actuation.

One must conclude that the sensor presented a nervous response because of the vibrations caused by the motors, this reflected on the control actuation, which has some variation. Aside from these facts, the controller led the pitch and roll to the setpoint with minimal control error considering the closed-loop system in hands.

4.2.1.2 Yaw controller

The yaw experiment was also conducted under the same conditions as the pitch and roll. The cables presented a permanent perturbation and the sensor has an abrupt value variation near the limits, that is, the sensor varies its value from 0º to 360º and back. In order to avoid this, the setpoint of the experiment was 40º and the permanent perturbation was around 3.5º. Figure 4.37 and Figure 4.38 represent the experimental result.

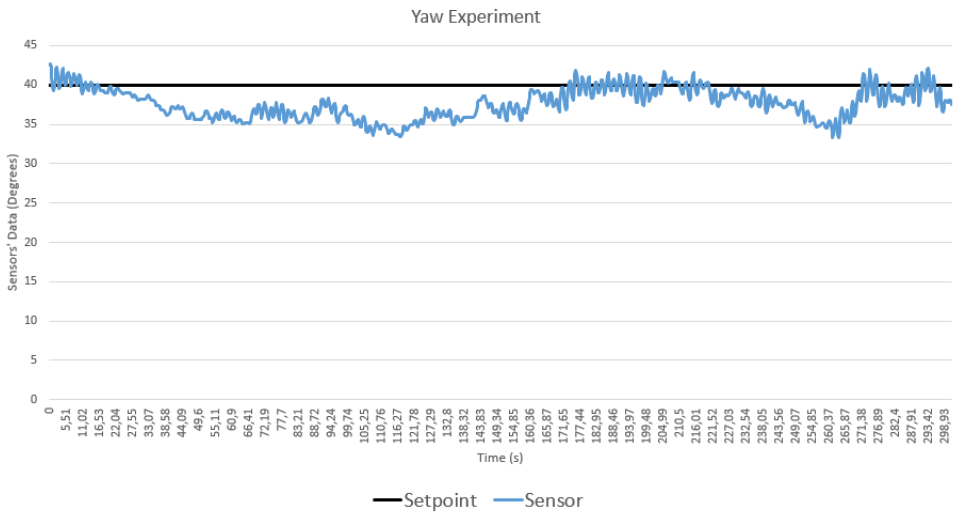


Figure 4.37 – Yaw sensor data received by the base station.

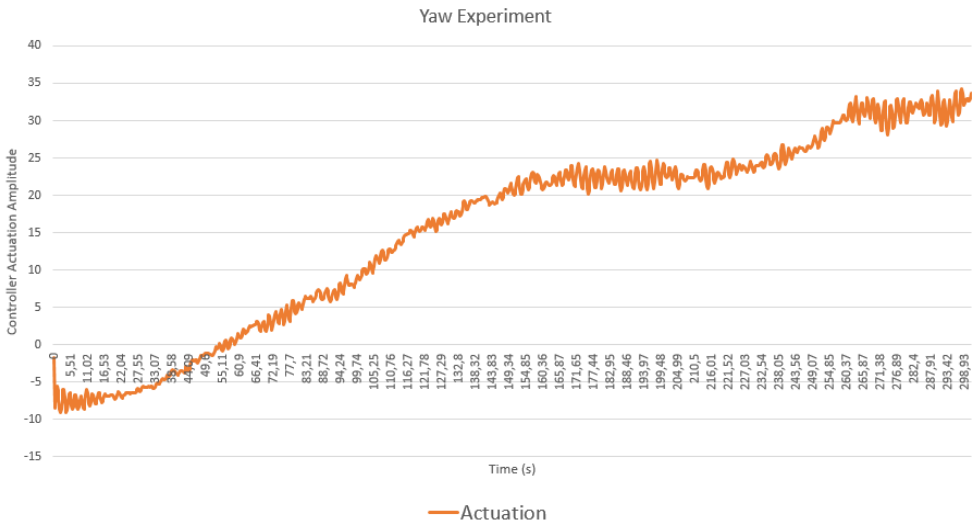


Figure 4.38 – Yaw PID control actuation.

The controller presented good performance leading the sensor value (the aircraft heading) towards the setpoint. Some actuation variation is also present as in the other experiments, one can also observe from the actuation that a deviation of the sensor is present; this is because the sensor has a slight decalibration due to the influence of the magnetic field generated by the motors. The quality of the sensors in these kinds of systems are imperative and the decalibration present was not predicted or expected from the sensor used.

4.2.1.3 Attitude control

The attitude control is the experiment containing all three controllers tested before. The result can be observed in Figure 4.39 and Figure 4.40.

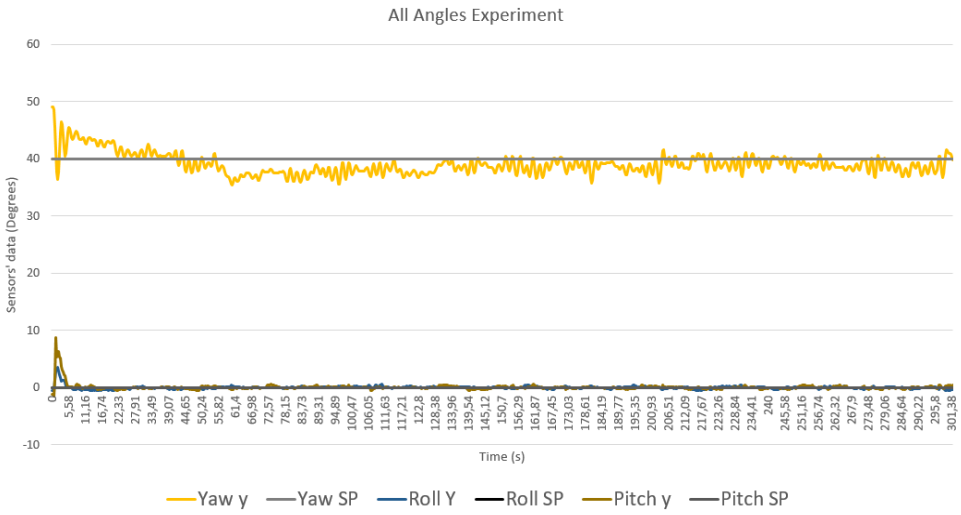


Figure 4.39 - Attitude sensors data received by the base station.

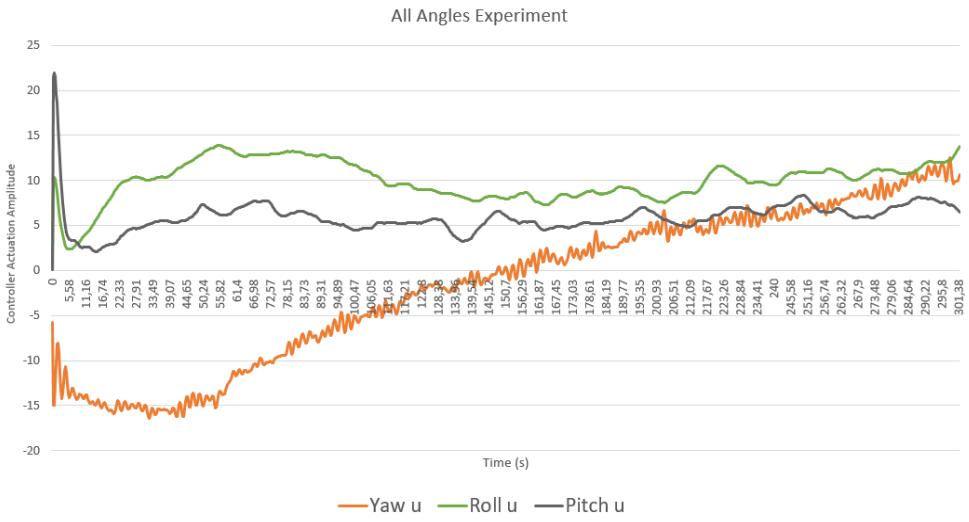


Figure 4.40 - Attitude PID controllers' actuation.

The controller presented good performance, considering the issue with the yaw sensor. It was very noticeable that the sensor decalibration led the actuator to the saturation and the cables did not allow the rotation of the aircraft. The sensorial fusion with other sensors is a complex topic for a possible future work; this is a conceivable way to solve the decalibration problem. Aside from this problem, the experiment in faulty condition was done.

4.2.2 Fault Tolerant Controller

Experiment under faulty conditions was made for the case of motor 1 failure. No more experiments were made because there was only one sensor used to measure current in the motors. The experiment can be observed in Figure 4.41 and Figure 4.42.

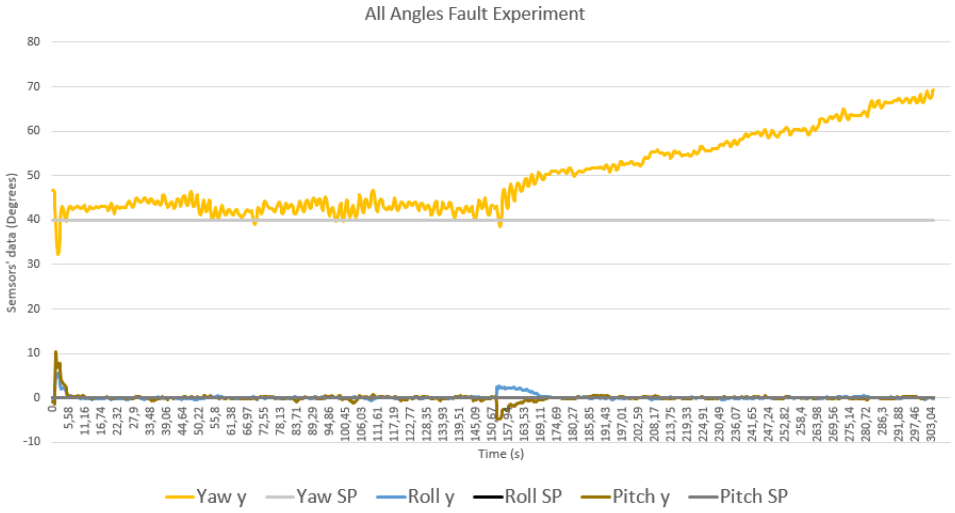


Figure 4.41 – Faulty attitude sensors data received by the base station.

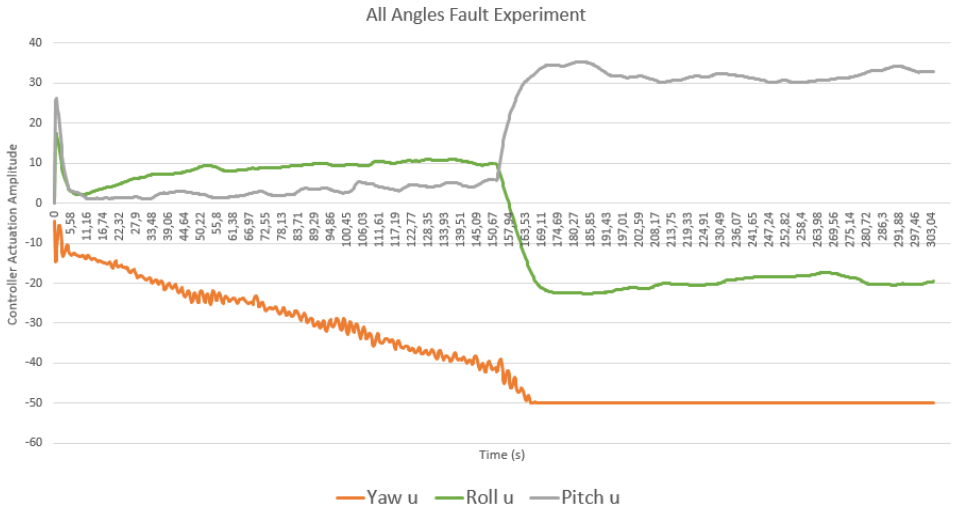


Figure 4.42 - Faulty attitude PID controllers' actuation.

The fault and reconfiguration occurred at around 150 s of the experiment. The controllers presented good performance and kept the stability of the attitude. The yaw was not able follow the setpoint because of the cables holding the aircraft and the already mentioned issue surrounding the decalibration.

4.3 Comparison and Results Analysis

The kinematic and dynamic models developed were fundamental to better understand the aircrafts behavior, allowing the design of fault tolerant control architectures, as well as the tuning of these controllers. The models implemented permitted the opened and closed loop control to be simulated before being implemented in the real aircraft.

The controllers designed using the Ziegler-Nichols and Åström Relay methods presented a good performance from the position to the angles controllers. The position and trajectory controller with yaw rotations needs more adjustments regarding the supervisor setpoint calculations.

The controllers designed using the PSO showed promising results. However, with the hardware and time available, the optimization algorithm presented a sub-optimal solution: some of the controllers found by the algorithm presented similar results to the previous methods. In order to obtain better results, the PSO needed more optimization time, with more particles and iterations to find better solutions.

Between the two control methodologies tested, the SMC revealed better performance regarding the components with strong non-linear behaviors (X and Y) than the PID. As mentioned *supra*, the SMC controller base is a non-linear design, allowing for the controller to better respond to non-linear systems like this. The PID, which is based on a linear design, presented satisfactory performance, assuring the stability of the closed-loop system. Table 4.1 and Table 4.2 present the control action variance and the mean-squared error between the two control architectures and according to all variables. The results presented correspond to the simulations based on the PSO, regarding the trajectory.

Table 4.1 - Control action variance comparison.

VAR	*X*	*Y*	*Z*	*Roll*	*Pitch*
PID	0.031	0.033	6.31E-4	1.26E-12	1.30E-12
SMC	0.026	0.025	0.0021	7.72E-11	7.70E-11

Table 4.2 – Means-squared error comparison.

MSE	*X*	*Y*	*Z*	*Roll*	*Pitch*
PID	0.0036	0.0035	3.14E-4	2.5E-5	2.51E-5
SMC	0.0027	0.0025	6.68E-4	3.59E-4	3.57E-4

Chapter 5

5 Conclusions and Future Work

5.1 General Conclusions

In this work, modeling and fault tolerant control of a quadcopter with X8 configuration was proposed. The main goal of this dissertation was to create a dynamic and kinematic model, in order to design fault tolerant controllers. This goal was fully achieved with a complete model of the aircraft, able to simulate faults / failures and design controllers capable of tolerating them.

Another objective was to assemble the quadcopter X8 design, in order to posteriorly apply the algorithms of fault tolerant control. Once again, the objective was accomplished with the assemble of a full operational aircraft with the intended architecture capable of flying under faulty conditions.

Moreover, an additional intent was to develop a virtual reality world where the model dynamics and kinematics could be easily observed and explained to people. Aside from the Simulink 3D not being a user-friendly software, the virtual reality world was created and the model was integrated in it. The virtual reality world has some performance issues due to the complexity of the calculations of the dynamic and kinematic model and, because of this, the world has low frame-rate.

The main contribution is the use of fault detection and diagnosis applied to a real world aircraft using algorithms of control allocations, in order to allow an over-actuated multirotor to fulfill a mission under faulty conditions. The fault detection and diagnosis to actuators was studied and validated in the simulation and implemented by means of the sensors available to the real world aircraft, which was able to maintain its attitude with faulty motors.

Even though the complexity of the open-loop system's dynamic and kinematic, the control architectures and the limitations of the hardware, the work was experimented in simulation and in the real world environment. The position and trajectory controllers were not implemented in the real world aircraft because of the inexistence of good security conditions to the aircraft and people. The altitude controller was also not tested due to the relation between signal to noise ration of the sensor, which is less than one in the existing test conditions. In addition, the base station was not developed, even though the communications from the aircraft side are prepared and working.

Part of the work of this dissertation contributed to a paper that was published in an international conference named "17th International Conference on Power Electronics and Motion Control" – PEMC 2016 in Varna Bulgaria (Brito, Brito Palma, Vieira Coito, & Valtchev, 2016). A tutorial session was also performed, result of a special invite from the conference organization committee.

5.2 Future Work

Although there has been an extensive work, some topics were left to future work due to the lack of time or hardware/software limitations. The following topics illustrate some of the possible future improvements to be made:

- There is work to be done in the trajectory using yaw, as well as the experiments with the real world aircraft. Furthermore, the energy efficiency could be improved, according to the trajectory;
- Sensorial data fusion. From one perspective, to correct the decalibration of the angles sensors and, on another perspective, to make the GPS information more precise and reliable;
- The lift off and landing are also complex maters and can be part of a future improvement, not only in static surfaces but also in motion surfaces;
- Fault detection and diagnosis of sensors is another possible upgrade to the current architecture with the implementations of observers for instance;
- Program the base station with a more user-friendly interface;
- Test other types of controllers;
- Investigate different performance indexes for control loop regarding drones;
- Implement control with acceleration and / or speed in the control loop.

6 References

Alves de Sousa, J. D. (2011). *Simulação e Desenvolvimento de um Veículo Aéreo Autónomo de Quatro Rotores*. MSc Thesis, Faculdade de Engenharia da Universidade do Porto.

Amazon Inc. (2016). Amazon PrimeAir. Retrieved March 1, 2016, from http://www.amazon.com/b?node=8037720011

Ang, K. H., Chong, G., & Li, Y. (2005). PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, *13*(4), 559–576. https://doi.org/10.1109/TCST.2005.847331

Åström, K. J., & Hägglund, T. (1995). PID controllers: theory, design and tuning. *Instrument Society of America*. https://doi.org/1556175167

Åström, K. J., & Hägglund, T. (2006). *Advanced PID Control*. https://doi.org/978-1-55617-942-6

Åström, K. J., & Wittænmark, B. (1994). Adaptive Control.

Åström, K. J., & Wittænmark, B. (1997). *Computer Control System - Theory and Design* (Third Edit). Prentice Hall. https://doi.org/10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C

Blanke, M., Kinnaert, M., Lunze, J., & Staroswiecki, M. (2006). *Diagnosis and Fault-Tolerant Control*. https://doi.org/10.1007/978-3-540-35653-0

Bouabdallah, S. (2007). *Design and Control of Quadrotors With Application To Autonomous Flying*. *Techniques*. Phd Thesis, École Polytechnique Fédérale de Lausanne. https://doi.org/10.5075/epfl-thesis-3727

Brito, V., Brito Palma, L. F. F., Vieira Coito, F., & Valtchev, S. (2016). Modeling and Supervisory Control of a Virtual X8-VB Quadcopter. In *17th International Conference on Power Electronics and Motion Control* (p. 8). Varna, Bulgária.

Brito Palma, L. F. F. (2007). *Fault detection, diagnosis and fault tolerance approaches in dynamic systems based on black-box models*. Phd Thesis, Universidade Nova de Lisboa - FCT - DEE.

Brito Palma, L. F. F., Vieira Coito, F., Gomes Ferreira, B., & Sousa Gil, P. (2015). PSO based on-line optimization for DC motor speed control. *9th International Conference on Compatibility and Power Electronics (CPE)*, 301--306. https://doi.org/10.1109/CPE.2015.7231090

Bryson, J. a. E. (1996). Optimal control-1950 to 1985. *IEEE Control Systems*, *16*(3), 26–33. https://doi.org/10.1109/37.506395

Cardoso, A. (2006). *Supervisão e Controlo de Sistemas Dinâmicos com Tolerância a Falhas – Contribuição para uma abordagem estruturada e robusta*. Phd Thesis, Faculdade de Ciências e Tecnologia - Universidade de Coimbra.

Convertawings Inc. (1956). Model A Quadrotor. *Flight*, 4762.

Crouch, T. D. (2004). *First Flight: The Wright Brothers and the Invention of the Airplane*. Retrieved from https://books.google.com/books?id=KbhcxujYZZsC&pgis=1 (2016-02-05)

DJI. (2016). Phantom 4. Retrieved January 13, 2016, from http://store.dji.com/new-release?from=store_index_banner

Dole, C. E., & Lewis, J. E. (2000). *Flight theory and aerodynamics: a practical guide for operational safety*. book, John Wiley & Sons.

Durham, W. C. (1993). Constrained control allocation. *Journal of Guidance Control and Dynamics*, *16*(4), 717–725. https://doi.org/10.2514/3.21072

Falkovich, G. (2011). Fluid mechanics: A short course for physicists.

Faraday, M. (1822). On Some New Electro-Magnetical Motion, and on the Theory of Magnetism. *Quarterly Journal of Science, Literature and the Arts*, *XII*, 74–96. Retrieved from http://archive.org/details/quarterlyjournal12jour (2016-02-02)

Federal Aviation Administration. (2008). *Pilot ' s Handbook of Aeronautical Knowledge*. *Pilot's Handbook of Aeronautical Knowledge*. Oklahoma. https://doi.org/10.1016/S0740-8315(86)80070-5

Ferreira, B. A. G. (2016). *Controlo e Diagnóstico de Falhas Baseado em Modo Deslizante*. MSc Thesis, Faculdade de Ciências e Tecnologias - Universidade Nova de Lisboa.

Flyablility SA. (2014). Gimball. Retrieved February 24, 2016, from http://www.flyability.com/applications/#inspec

Frank, P. M. (1996). Analytical and Qualitative Model-based Fault Diagnosis – A Survey and Some New Results. *European Journal of Control*, *2*(1), 6–28. https://doi.org/10.1016/S0947-3580(96)70024-9

Hägglund, T., & Aström, K. J. (2000). Supervision of adaptive control algorithms. *Automatica*, *36*(8), 1171–1180. https://doi.org/10.1016/S0005-1098(00)00026-1

Hahn, H. (2002). *Rigid Body Dynamics of Mechanisms: Theoretical basis*. Springer Science & Business Media. Retrieved from https://books.google.com/books?id=MqrN3KY7o6MC&pgis=1 (2016-02-20)

Heatly, M. (1986). *Illustrated History of Helicopters*.

Hespanha, J. (2001). Tutorial on supervisory control. *Lecture Notes for the Workshop Control Using Logic and Switching for the 40th Conf. on Decision and Contr.*, 1–46.

Hespanha, J. P., Liberzon, D., & Stephen Morse, A. (2003). Center for control, dynamical systems, and computation. *Nonlinearity*, (C), 1171–1171.

Kaempffert, W. (1911). *The new art of flying*. Retrieved from https://archive.org/stream/newartflying00kaemgoog#page/n12/mode/2up (2016-02-05)

Keele, K. D. (2014). *Leonardo Da Vinci's Elements of the Science of Man*. Retrieved from https://books.google.com/books?hl=pt-PT&lr=&id=Hkm0BQAAQBAJ&pgis=1 (2016-02-04)

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the 1995 IEEE International Conference on Neural Networks. Part 1 (of 6)*, *4*, 1942–1948. https://doi.org/10.1007/s11721-007-0002-0

Levine, W. S. (2000). *The Control Handbook*.

M.B. Armand, J.M. Chabagno, M. D. (1978). *Extended Abstracts*. St. Andrews, Scotland: Second International Meeting on Solid Electrolytes. Retrieved from https://books.google.pt/books/about/Second_International_Meeting_on_Solid_El.html?id=GXTftgAACAAJ&pgis=1

Manuel Stephan, A., & Nahm, K. S. (2006). Review on composite polymer electrolytes for lithium batteries. *Polymer*, *47*(16), 5952–5964. https://doi.org/10.1016/j.polymer.2006.05.069

McFarlane, D., & Glover, K. (1992). A loop-shaping design procedure using H/sub infinity / synthesis. *IEEE Transactions on Automatic Control*, *37*(6), 759–769. https://doi.org/10.1109/9.256330

Mellinger, D., Michael, N., & Kumar, V. (2012). Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, *0*(0), 1–11. https://doi.org/10.1177/0278364911434236

Momont, A. (2014). Drones for Good. Retrieved December 12, 2016, from http://www.alecmomont.com/projects/dronesforgood

Murata, K., Izuchi, S., & Yoshihisa, Y. (2000). An overview of the research and development of solid polymer electrolyte batteries. *Electrochimica Acta*, *45*(8–9), 1501–1508. https://doi.org/10.1016/S0013-4686(99)00365-5

Oehmichen, E. (1924). A successful french helicopter. *Flight*.

Oppenheimer, M. W., Doman, D. B., & Bolender, M. A. (2006). Control allocation for over-actuated systems. In *14th Mediterranean Conference on Control and Automation, MED'06*. https://doi.org/10.1109/MED.2006.328750

Pereira da Costa, S. E. A. (2008). *Controlo e Simulação de um Quadrirotor convencional*. MSc Thesis, Instituto Superior Técnico da Universidade Técnica de Lisboa.

Qadir, A. F. M. S. (2013). *Electro-Mechanical Modeling of SEDM (Separately Excited DC Motor) & Performance Improvement Using Different Industril Controllers*. Lulu.com. Retrieved from https://books.google.com/books?id=8_SgBQAAQBAJ&pgis=1 (2016-02-02)

Rondeau, L., Ruelas, R., Levrat, L., & Lamotte, M. (1997). A defuzzification method respecting the fuzzification. *Fuzzy Sets and Systems*, *86*(3), 311–320. https://doi.org/10.1016/S0165-0114(95)00399-1

Ross, T. J. (2009). *Fuzzy Logic with Engineering Applications*. Retrieved from https://books.google.com/books?hl=pt-PT&lr=&id=nhz1f9j6_SMC&pgis=1 (2016-02-11)

Royal Institution of Great Britain. (1822). Quarterly Journal of Science, Literature and the Arts., *Vol XII*, 492. Retrieved from http://www.biodiversitylibrary.org/item/50623 (2016-02-02)

Safonov, M. G., & Fan, M. K. H. (1997). Editorial, Special Issue on Multivariable Stability Margin. *International Journal of Robust and Nonlinear Control*, *7*, 97–103.

Santina, M. S., & Stubberud, A. R. (2005). Basics of Sampling and Quantization. In *Handbook of Networked and Embedded Control Systems* (pp. 45–70). Boston, MA: Birkhäuser Boston. https://doi.org/10.1007/0-8176-4404-0_3

Santos, T. M. A. (2016). *Fuzzy Data Fusion Approach for Remote Sensing Classification*. MSc Thesis, Universidade Nova de Lisboa - FCT.

Scrosati, B. (2001). Progress in lithium polymer battery R&D. *Journal of Power Sources*, *100*(1–2), 93–100. https://doi.org/10.1016/S0378-7753(01)00886-2

Simoes, M. G. (2003). Introduction to Fuzzy Control ∗. *Colorado School of Mines, Engineering Division, Golden, Colorado*, 1–5. https://doi.org/10.1177/0094582X0202900101

Spurgeon, S. K., Sabanovic, A., & Fridman, L. (2004). *Variable Structure Systems from principles to implementation*.

Steffen, T. (2005). *Control Reconfiguration of Dynamical Systems: Linear Approaches and Structural Tests*. Retrieved from https://books.google.com/books?hl=pt-PT&lr=&id=t7jJQu2RaXIC&pgis=1 (2016-02-16)

Utkin, V. (1992). *Sliding Modes in Control and Optimization*. *SciencesNew York*. https://doi.org/10.1007/978-3-642-84379-2

Utkin, V. (2005). *Sliding Mode Control*. *Variable structure systems: from principles to*. Retrieved from https://www.google.com/books?hl=pt-PT&lr=&id=zuD6Mh38KbYC&oi=fnd&pg=PA3&dq=Sliding+mode+control&ots=-aunBq7odL&sig=caiYbHaWTpnL7QSPeh05-HAeJhc (2016-09-15)

Webb, C. J., Budman, H. M., & Morari, M. (1995). Identification of Uncertainty Bounds for Robust Control with Applications to a Fixed Bed Reactor. *Industrial & Engineering Chemistry Research*, *34*(5), 1743–1754. https://doi.org/10.1021/ie00044a026

Welch, G., & Bishop, G. (2006). An Introduction to the Kalman Filter. *In Practice*, *7*(1), 1–16. https://doi.org/10.1.1.117.6808

Wilson, T. G., & Trickey, P. H. (1962). D-C machine with solid-state commutation. *Electrical Engineering*, *81*(11), 879–884. https://doi.org/10.1109/EE.1962.6446586

Yedamale, P. (2003). Brushless DC (BLDC) motor fundamentals. *Microchip Technology Inc*, 1–20. https://doi.org/10.1093/bioinformatics/btr584

Young, W. R. (1982). *The Helicopters*. *The Epic of Flight*. Chicago: Time-Life Books.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353. https://doi.org/10.1016/S0019-9958(65)90241-X

Zhou, K. (1999). *Essentials of Robust Control*. https://doi.org/10.1177/014662168400800314

Ziegler, J., & Nichols, N. (1942). Optimum settings for automatic controllers. *Trans. ASME*. Retrieved from http://staff.guilan.ac.ir/staff/users/chaibakhsh/fckeditor_repo/file/documents/Optimum Settings for Automatic Controllers (Ziegler and Nichols, 1942).pdf (2016-07-05)

Attachments

A. Arduino Code

I. Initialization

Initialization of the Absolute Orientation Sensor and GPS.

```
void initSens() {
  Serial2.print("Initializing Sensors.");
  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  //GPS.begin(9600);
  mySerial.begin(9600); Serial2.print(".");
  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY); Serial2.print("..");
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); Serial2.print("..");   // 1 Hz update rate
  GPS.sendCommand(PGCMD_NOANTENNA); Serial2.print(".");
  //Sensor Initialization
  Sensor.initSensor(); Serial2.print(".");//The I2C Address can be changed here inside this function in the library
  Sensor.setOperationMode(OPERATION_MODE_NDOF); Serial2.print(".");//Can be configured to other operation modes as desired
  Sensor.setUpdateMode(MANUAL);       Serial2.print(".");//The default is AUTO. Changing to manual requires calling the relevant
                                                         // update functions prior to calling the read functions
  Sensor.updateAccelConfig();//Setting to MANUAL requires lesser reads to the sensor

  Shield_offset[1] =  Sensor.readEulerRoll();
  Shield_offset[2] = Sensor.readEulerPitch();
  Shield_offset[3] = Sensor.readEulerHeading();
}
```

II. Structures Used

```
struct Actuator {
  int pin;
  Servo u;
};

struct ctrPID {
  float kp;  float ti;  float td;
  float N; float Tt; float bi;
  float ad; float bd; float ao;
  float yold; float v; float e;
  float P; float I; float D;
  float uhigh; float ulow;
  float sp;
  float u;
};
struct ctrSMC {
  float e; float eold; float ge;
  float pc; float gaw; float c;
  float lambda; float ec;
  float de; float deold; float ie;
  float ieold; float oc; float rhoc;
  float uhigh; float ulow;
  float sp;
  float v; float vold;
  float u; float uold;
};

struct ctrRelay {
  float threshold;
  float u;
};

struct Sensor_read {
  float xx;  float yy;  float zz;
  float roll;  float pitch;  float yaw; float yawold;
};

struct Radio {
  int pin;
  double input;
};
```

III. Communications

```
void Comm(int ctrflag) {
  Serial2.print("$");
  float time = (millis()-tcomm) / 1000.00;
  Serial2.print(time);Serial2.print(";");
  //ctrflag = 1 - PID
  //        = 2 - SMC
  if (ctrflag == 1) {
    Serial2.print(ctryaw.sp); Serial2.print(";");   Serial.print(ctryaw.sp); Serial.print(" ");
    Serial2.print(ctryaw.u); Serial2.print(";");    Serial.print(ctryaw.u); Serial.print(" ");
    Serial2.print(Sensors.yaw); Serial2.print(";"); Serial.print(Sensors.yaw); Serial.print(" ");

    Serial2.print(ctrroll.sp); Serial2.print(";");   Serial.print(ctrroll.sp); Serial.print(" ");
    Serial2.print(ctrroll.u); Serial2.print(";");    Serial.print(ctrroll.u); Serial.print(" ");
    Serial2.print(Sensors.roll); Serial2.print(";"); Serial.print(Sensors.roll); Serial.print(" ");

    Serial2.print(ctrpitch.sp); Serial2.print(";");  Serial.print(ctrpitch.sp); Serial.print(" ");
    Serial2.print(ctrpitch.u); Serial2.print(";");   Serial.print(ctrpitch.u); Serial.print(" ");
    Serial2.print(Sensors.pitch); Serial2.print(";");Serial.print(Sensors.pitch); Serial.print(" ");

    Serial2.print(ctralt.sp); Serial2.print(";");    Serial.print(ctralt.sp); Serial.print(" ");
    Serial2.print(ctralt.u); Serial2.print(";");     Serial.print(ctralt.u); Serial.print(" ");
    Serial2.print(Sensors.zz); Serial2.print(";"); Serial.print(Sensors.zz); Serial.print("  ");

    Serial2.print(Sensors.motor[0]);Serial2.print(";");Serial2.print(Sensors.motor[1]);Serial2.print(";");Serial2.print(Sensors.motor[2]);Serial2.print(";");
    Serial2.print(Sensors.motor[3]);Serial2.print(";");Serial2.print(Sensors.motor[4]);Serial2.print(";");Serial2.print(Sensors.motor[5]);Serial2.print(";");
    Serial2.print(Sensors.motor[6]);Serial2.print(";");Serial2.println(Sensors.motor[7]);

  }
```

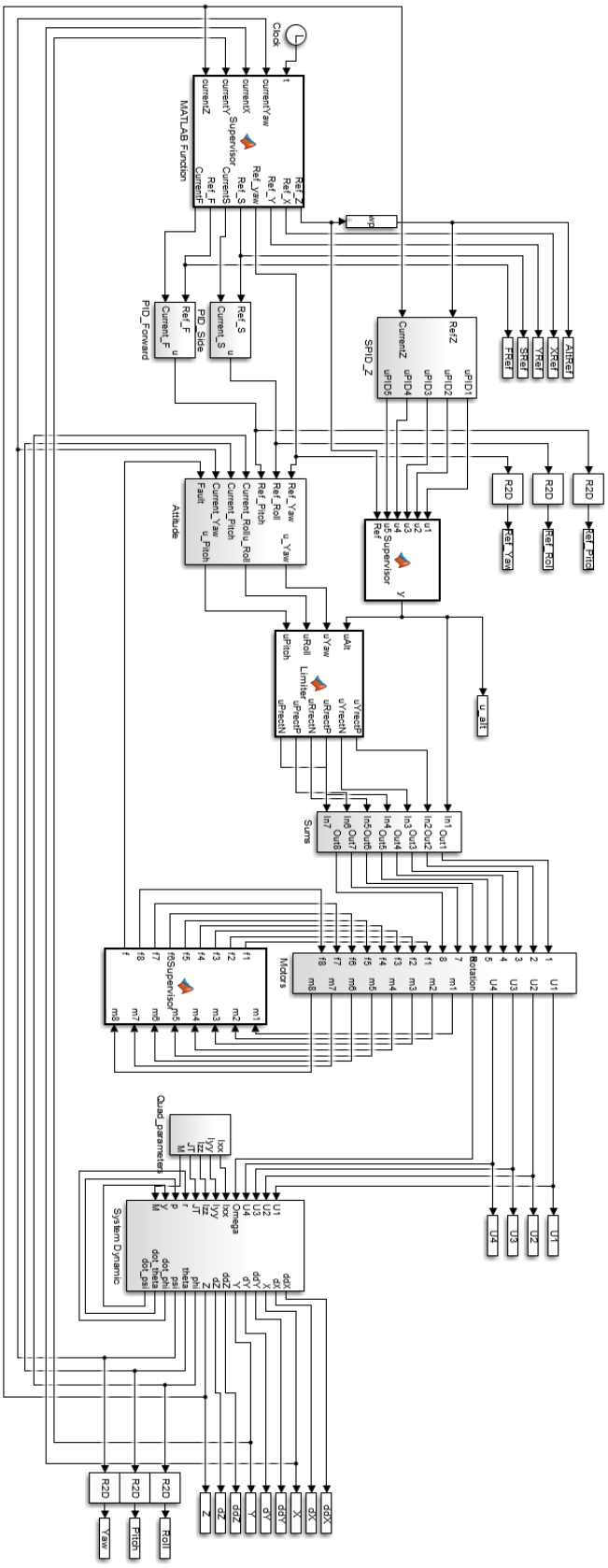
IV. Data Logging

```
void data_logger() {
  unsigned long currentMillis = millis();

  if (GPS.newNMEAreceived()) {

    if (GPS.parse(GPS.lastNMEA()))  // this also sets the newNMEAreceived() flag to false
    {
      if (datFile)
      {
        gpsFile.println(GPS.lastNMEA());
        gpsFile.flush();
      }
    }
  }
  if (currentMillis - DataMillis >= DataInterval)
  {
    DataMillis = currentMillis;

    data = "";
    data.concat(ctryaw.sp);
    data.concat(" ; ");
    data.concat(ctryaw.u);
    data.concat(" ; ");
    data.concat(Sensors.yaw);
    data.concat(" ; ");
    data.concat(ctrroll.sp);
    data.concat(" ; ");
    data.concat(ctrroll.u);
    data.concat(" ; ");
    data.concat(Sensors.roll);
    data.concat(" ; ");
    data.concat(ctrpitch.sp);
    data.concat(" ; ");
    data.concat(ctrpitch.u);
    data.concat(" ; ");
    data.concat(Sensors.pitch);
    data.concat(" ; ");
    data.concat(ctralt.sp);
    data.concat(" ; ");
    data.concat(ctralt.u);
    data.concat(" ; ");
    data.concat(Sensors.zz);
    data.concat(" ;  ");
    data.concat(Sensors.motor[0]);data.concat(" ; ");data.concat(Sensors.motor[1]);data.concat(" ; ");data.concat(Sensors.motor[2]);data.concat(" ; ");
    data.concat(Sensors.motor[3]);data.concat(" ; ");data.concat(Sensors.motor[4]);data.concat(" ; ");data.concat(Sensors.motor[5]);data.concat(" ; ");
    data.concat(Sensors.motor[6]);data.concat(" ; ");data.concat(Sensors.motor[7]);
    data.concat("\t");

    if (datFile)
    {
      datFile.println(data);
      datFile.flush();
    }
  }
}
```

V. Supervisor

```
void FaultDetect() {
  Sensors.motor[0] = -(analogRead(A0) - 761) / 400 * 20;
  Sensors.motor[1] = -(analogRead(A1) - 761) / 400 * 20;
  Sensors.motor[2] = -(analogRead(A2) - 761) / 400 * 20;
  Sensors.motor[3] = -(analogRead(A3) - 761) / 400 * 20;
  Sensors.motor[4] = -(analogRead(A4) - 761) / 400 * 20;
  Sensors.motor[5] = -(analogRead(A5) - 761) / 400 * 20;
  Sensors.motor[6] = -(analogRead(A6) - 761) / 400 * 20;
  Sensors.motor[7] = -(analogRead(A7) - 761) / 400 * 20;

  if (Sensors.motor[0] == 0.00) {
    Serial2.println("Motor 1 fault!"); Serial.println("Motor 1 fault!");
    if (F[2] == 0){
      F[4] = 1; F[1] = 0; F[3] = 0;
    }else
      F[6] = 0;
  }
  if (Sensors.motor[1] == 0.00) {
    Serial2.println("Motor 2 fault!"); Serial.println("Motor 2 fault!");
    if (F[3] == 0){
      F[7] = 1; F[0] = 0; F[2] = 0;
    }else
      F[7] = 0;
  }
  if (Sensors.motor[2] == 0.00) {
    Serial2.println("Motor 3 fault!"); Serial.println("Motor 3 fault!");
    if (F[0] == 0){
      F[6] = 1; F[1] = 0; F[3] = 0;
    }else
      F[4] = 0;
  }
  if (Sensors.motor[3] == 0.00) {
    Serial2.println("Motor 4 fault!"); Serial.println("Motor 4 fault!");
    if (F[1] == 0){
      F[7] = 1; F[0] = 0; F[2] = 0;
    }else
      F[5] = 0;
  }//////////////////////////////////////////////////////////////////////////
  if (Sensors.motor[4] == 0.00) {
    Serial2.println("Motor 5 fault!"); Serial.println("Motor 5 fault!");
    if (F[6] == 0){
      F[4] = 1; F[1] = 0; F[3] = 0;
    }else
      F[2] = 0;
  }
  if (Sensors.motor[5] == 0.00) {
    Serial2.println("Motor 6 fault!"); Serial.println("Motor 6 fault!");
    if (F[7] == 0){
      F[1] = 1; F[4] = 0; F[6] = 0;
    }else
      F[3] = 0;
  }
  if (Sensors.motor[6] == 0.00) {
    Serial2.println("Motor 7 fault!"); Serial.println("Motor 7 fault!");
    if (F[4] == 0){
      F[2] = 1; F[5] = 0; F[7] = 0;
    }else
      F[0] = 0;
  }
  if (Sensors.motor[7] == 0.00) {
    Serial2.println("Motor 8 fault!"); Serial.println("Motor 8 fault!");
    if (F[5] == 0){
      F[3] = 1; F[4] = 0; F[6] = 0;
    }else
      F[1] = 0;
  }
}
```

B. Simulink Model



C. Virtual World